既存メタデータスキーマを用いたメタデータインスタンスからのスキーマ推定手法

既存メタデータスキーマを用いた
メタデータインスタンスからのスキーマ推定手法

筑波大学
図書館情報メディア研究科
2014 年 3 月
田中　圭

目次

図目次

表目次

1. はじめに

World Wide Web(WWW)上でデータを共有するLinked Open Data(以下LOD)という試みが注目され、2009年以降公開されているデータセット数が急増している[1][2]。LODとして公開されているデータセットには政府が公開しているものをはじめとして地理情報やメディア情報など様々な分野のものが存在し、異なる分野間のデータをマッシュアップする(組み合わせる)ことが可能となっている。しかし、LODデータセットから利用者が目的のデータを取得するためには、各LODデータセットに蓄積されているメタデータの構造を理解し、問い合わせを行う必要がある。メタデータスキーマは、どのような語彙を用いてどのような構造を持つメタデータを記述するのか、また、メタデータ記述項目にどのような制約条件を与えるのかといった仕様のことで、これを定めることで明確な規則に従ったメタデータの記述が可能となる。そして、LODデータセットのメタデータスキーマが公開されていれば、LOD利用者がデータセットに蓄積されているメタデータの構造を理解する手間が解消される。しかし、メタデータスキーマを公開しているLODデータセットはまだ少ない[3]。そのため、利用者はLODデータセットのデータ構造の理解に手間がかかり、目的のデータを取得することが困難になっている。

本研究ではLOD利用者がデータセットのデータ構造を理解することを目的として、既存メタデータスキーマを用いたメタデータインスタンスからのスキーマ推定手法の提案と、手法に基づいたシステムを構築した。本手法は、LODデータセット中のメタデータインスタンスから使用されているクラスとプロパティを抽出し、公開されているメタデータ語彙定義からクラスとプロパティの語彙定義及び不足している情報を取得し、スキーマの構成要素を推定することでスキーマを推定する。

本論文では、まず2章でLODの特徴と利用例について述べ、構造化問い合わせ言語SPARQLを用いたLODデータセットへの問い合わせについて説明する。次に3章でメタデータスキーマの役割、問題点について述べ、4章で解決手法としてメタデータスキーマ推定手法を述べる。5章では4章で提案した手法の実現について述べ、6章で評価実験、7章で考察と課題を述べる。8章で本研究と関連する研究を紹介し、最後に9章でまとめを述べる。

なお、本研究の手法で推定したスキーマは、メタデータ情報基盤構築事業のメタデータ情報共有のためのガイドラインで提案されている簡易DSPの形式を基にしている[4][5]。

2. Linked Open Data の利用

本章では本研究で扱う Linked Open Data(LOD)の特徴と、LOD を利用した Web アプリケーションの紹介、LOD を利用するための構造化問い合わせ言語 SPARQL について述べる。

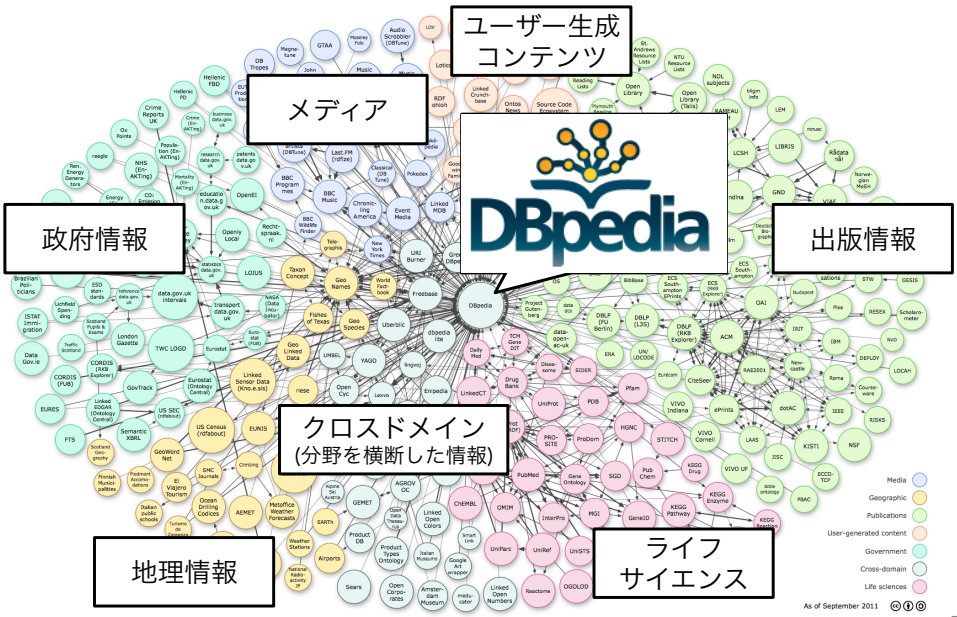
2.1. Linked Open Data

初期の WWW は HTML 文書同士がハイパーリンクによってリンクされている「文書のウェブ(Web of Documents)」であった。文書のウェブは人間が記述し、人間が読むための Web として普及したが、HTML のハイパーリンクにはリンクされているドキュメント間の関係を示す機械可読な情報が含まれていない。そのため、リンクの意味は人間が読むまで解釈されない[6]。これに対し、WWW 上の情報の意味を機械が理解し、人間が情報を探索したり利用したりすることを支援できるような世界を目的として 1998 年に「Semantic Web」が提唱された[7]。その実現に向けた試みの中で、Tim Berners-Lee が提唱した 4 つの原則（Linked Data の基本原則）を基に 2006 年頃より進められている活動が「Linked Data」である。以下は Linked Data の基本原則である[30]。

1. *Use URIs as names for thins. 「あらゆる事物に URI を付与すること」*
2. *Use HTTP URIs so that people can look up those names.*
   「誰でも事物の内容が確認できるように、URI は HTTP 経由で参照できること」
3. *When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).*
   「URI を参照したときは、標準の技術(RDF や SPARQL 等)を使用して関係する有用な情報を利用できるようにすること」
4. *Include links to other URIs. so that they can discover more things.*
   「より多くの事物を発見できるように、他の URI へのリンクを含めること」

Linked Data は、様々な分野の情報資源を集約、関連付けし、利用できるようにする活動で、ある情報について記述したメタデータの中に他の情報資源との関連を機械にとって意味のあるリンクで結びつけている[1]。

Linked Data では必ずしも誰もがデータを自由に利用できる状態で公開する必要はなく、メタデータ作成者が所属する組織内でのみ利用可能な内部的データとしても利用されている。これに対し、特にメタデータを誰でも自由に利用できる状態で公開している Linked Data を「Linked Open Data」（以下 LOD)と呼ぶ[1][2]。LOD として公開しているデータセット間の相互関係を図示したものが LOD クラウドである(図 1,2 参照)。LOD クラウドは Wikipedia を LOD 化した DBpedia[8]を中心として、書誌情報や地理情報、メディア、政府が公開する公共情報など、様々な分野から公開され、2009 年以降急激に大きくなっている。そうした LOD データセットの増加に伴い、最近では Web アプリケーションに利用するデータとして LOD が用いられるケースも増加している。[2][9]。



50

図 1 LOD クラウド（引用文献[2]）

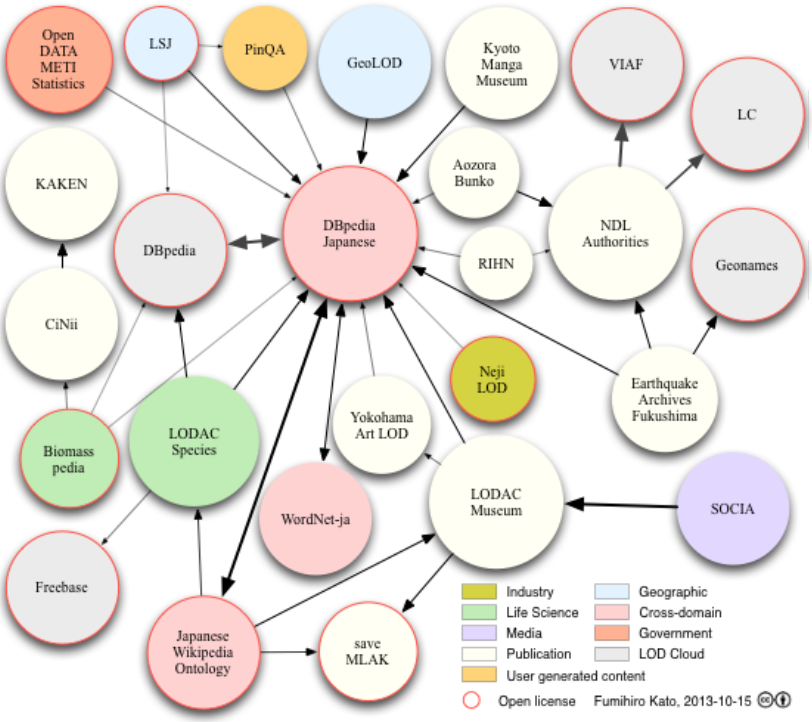


図 2 日本語版 LOD クラウド（引用文献[9]）

2.2. LOD を利用した Web アプリケーションの例

LOD の特徴は(i)様々な分野のデータが公開されていること(ii)メタデータを誰でも自由に取得・編集ができることである。本節では(i)の例として 1)Yokohama Art Spot を、(ii)の例として 2)Where Does My Money Go?日本語版について述べる。

1) Yokohama Art Spot

Yokohama Art Spot[10]とは、公益財団法人横浜市芸術文化振興財団が公開しているヨコハマ・アート LOD(Yokohama Art LOD)[11]を用いて、横浜市内の芸術関連施設とそこで開催されるイベントを案内する Web サイトである。スタート画面には横浜市内の地図が表示され、施設及びイベント情報がマッピングされる。この地図上には位置情報 Q&A サービス PinQA[12]で公開されている横浜周辺の Q&A 情報も同時に表示される。そして、一部の美術館に対して国内博物館情報の Linked Data である LODAC Museum[13]とリンクしており、所蔵する作品をジャンルごとに閲覧することもできる。

Yokohama Art Spot はブラウザ上で表示している地図上の地理情報から「PinQA の Q&A 情報」、「ヨコハマ・アート LOD の施設情報(一部 LODAC Museum から)、イベント案内」を取得し地図上にマッピング(視覚化)している。(図 3 参照)このようなアプリケーションの作成では各データセットから作成者が結びつけたいデータがどのような構造で蓄積されているかを把握して結びつけている。図 3 右上の日本語版 LOD クラウドにあるように、PinQA とヨコハマ・アート LOD は直接リンクがつながっていないが、DBpedia Japanese を介してつながっている。「博物館の情報」と「人々の Q&A」を結びつけ、地図上にマッピングすることで、各施設がシームレスに連携して情報を共有できる。このように、異なる分野のデータがリンクされていることでデータに新たな価値が発見できるのも LOD の特徴である。

図 3 Yokohama Art Spot(左下) 利用データセットの関係図(右上)

2) Where Does My Money Go? 日本語版

Where Does My Money Go? 日本語版[14]は、利用者の年間収入を入力すると、納めた税金が一日あたりどう使われているかを知ることができる Web サービスである。このサービスでは自治体が提供している主要事業の予算データを LOD 化し利用しており、β版では横浜市の市税を対象として構築されていたが[15]、Open Knowledge Foundation Japan のプロジェクトとして成長し、2014 年 1 月時点で 69 自治体の Web サイトを立ち上げている。

Where Does My Money Go? では総務省が公開している目的別歳出内訳を市町村ごとのプロジェクト参加者が LOD 化し、それぞれの自治体が公開している市税額の配分比率などを用いて図 4 の様にサイト利用者が選択した年収から税金が一日あたり何に使われているかを表示する。

このプロジェクトは 2014 年 1 月現在も参加者を募っており、各自治体の予算データを有志が LOD 化して Web サイトとして公開することで参加自治体が増加している。プロジェクト参加への手間を省くために公式サイトではサイト立ち上げのフローを提示しており、メタデータの作成手順も詳しく紹介されている。これはデータ公開者(団体)がデータセットのデータ構造を詳細に公開している例である。

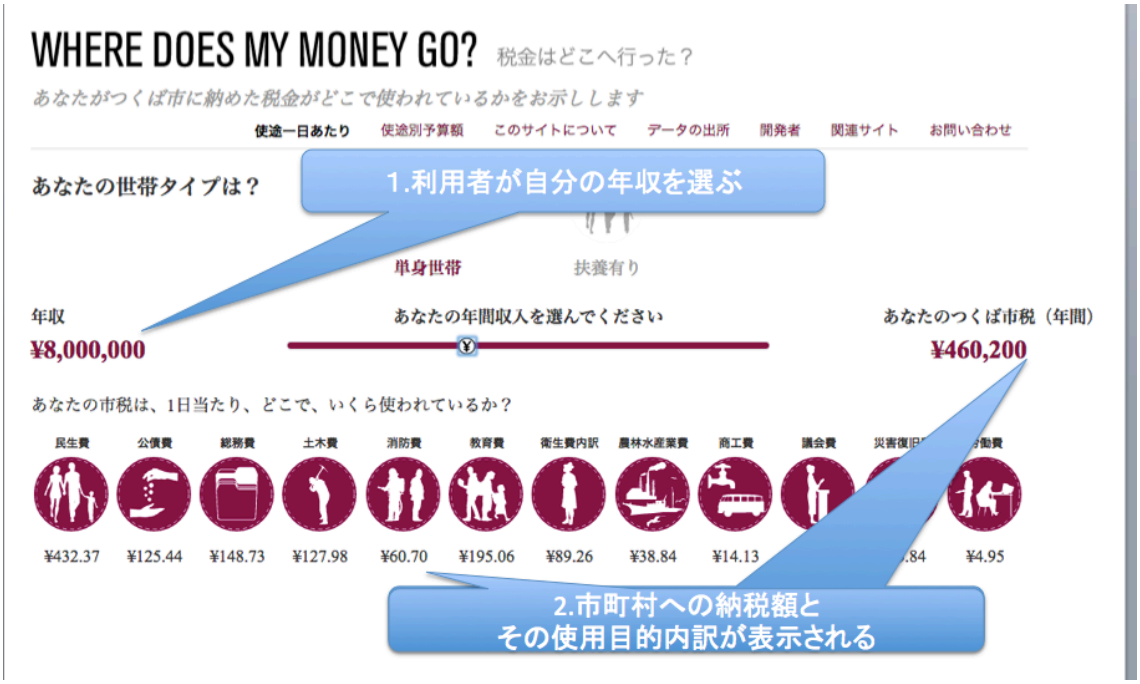


図 4 Where Does My Money Go?（つくば市）

2.3. SPARQL による構造化問い合わせ

LOD を利用した Web アプリケーションを作成するためには、必要なデータを問い合わせて取得する方法が必要である。データ問い合わせのための標準的な方法が、対象のデータセットのグラフ構造を把握した上で RDF クエリ言語 SPARQL を用いた LOD データセットへの構造化問い合わせを行う方法である。

Resource Description Framework

LOD では公開するメタデータを Resource Description Framework(RDF)[16]で記述する。RDF とは、リソースに関する情報を主語（リソース)・述語（プロパティ)・目的語（プロパティの値）のトリプルで明瞭かつ論理的に表現するデータモデルである[17]。

RDF トリプルの集合を RDF グラフと呼ぶ。RDF グラフは様々な記述方法によって表すことができる。例えば、XML を用いた RDF/XML[40]、主語・述語・目的語の URI を<>で囲んで記述し、順番に並べる N-Triples[41]、接頭辞を用いた表記や N-Triples にはない省略記法がある Turtle[42]などがある。

「URI “http://example.com/person01” で識別される Person クラスのインスタンスの名前は田中圭である」

RDF/XML

```
<foaf:Person rdf:about="http://example.com/person01">
  <foaf:name>田中圭</foaf:name>
</foaf:Person>
```

N-Triples

```
<http://example.com/person01> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person>.
<http://example.com/person01> < http://xmlns.com/foaf/0.1/name> “田中圭”.
```

Turtle

```
@prefix ex: <http://example.com/> .
ex:person01 rdf:type foaf:Person;
        foaf:name “田中圭”.
```

図 5 RDF の記述方法の一例(参考文献[17])

図 6 は有向グラフを用いて「URI “http://example.com/person01” で識別される Person クラスのインスタンスの名前は田中圭である」というメタデータを表現した RDF グラフの例である。図中の楕円はリソースを表し、矢印はプロパティを表している。 目的語がリテラルの場合、楕円ではなく長方形で表す。RDF グラフではノードは目的語であると同時に別のトリプルの主語にもなる。(図 7 参照)

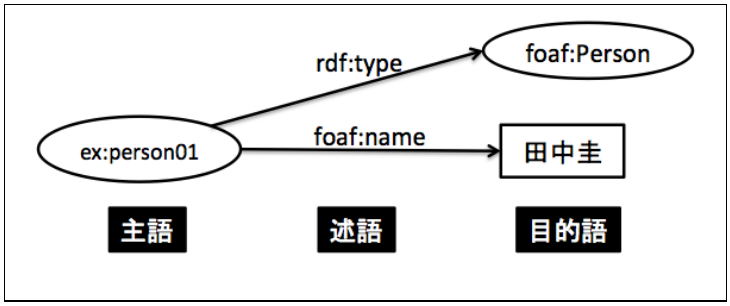


図 6 RDF グラフ　表現例

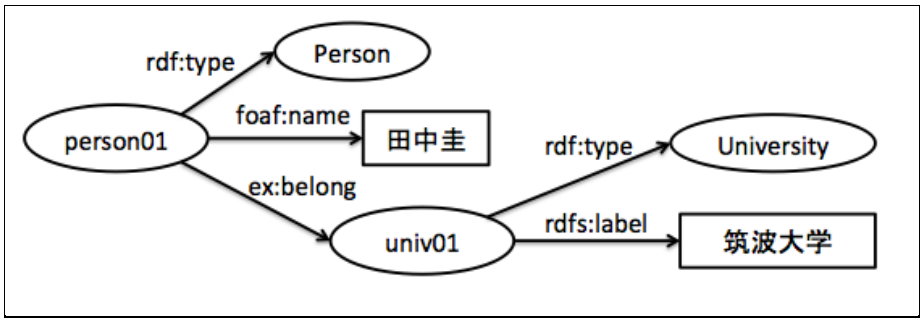


図 7 RDF トリプルが連結している例

SPARQL

RDF グラフに対するデータ問い合わせは、RDF クエリ言語 SPARQL[21][22]が用いられる。グラフパターン(構造)をクエリとして記述し、グラフパターンに一致するデータの検索を行う[23]。

以下の例は、与えられたリソース<http://example.com/book/book1>につけられているタイトル(http://purl.org/dc/elements/1.1/title)を発見するための SPARQL クエリである。

RDF データ(N-Triples 表記)

```
<http://example.com/book/book1> <http://purl.org/dc/elements/1.1/title> “羅生門”.
<http://example.com/book/book1> <http://purl.org/dc/elements/1.1/creator> “太宰治”.
<http://example.com/book/book2> <http://purl.org/dc/elements/1.1/title> “坊っちゃん”.
<http://example.com/book/book2> <http://purl.org/dc/elements/1.1/creator> “夏目漱石”
```

SPARQL クエリ

```
SELECT ?title
WHERE {
  <http://example.com/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

WHERE 句の中で記述されているグラフパターンとマッチする RDF トリプルの、SELECT 句で指定したインスタンスが結果として出力される。

title
羅生門
坊っちゃん

RDF で記述したデータは、Virtuoso[18]や Sesame[19]などの RDF 用データベースソフトウェア(トリプルストア)に蓄積され、SPARQL Endpoint[20]と呼ばれる HTTP 経由で SPARQL 問い合わせを受けつけ検索結果を返すサービスを介し、LOD データセットとして公開される。

SPARQL Endpoint には Web ブラウザをインタフェースとしているものもある。図 8 は落語家の名跡変化や落語家間の名跡の継承などを LOD 化した「落語家 LOD」[24]を公開している SPARQL Endpoint に SPARQL クエリを実行している様子である。クエリを実行すると図の右下のように問い合わせ結果が出力される。図の左上にあるクエリは「落語家に関する情報（氏名や生年月日、来歴情報など全て)」を発見するためのものである。しかしながら、このようなクエリを記述するには検索対象のデータセットに格納されている落語家として識別されるリソースが rakugo:Rakugoka クラスのインスタンスとして記述されていることを把握していなければならない。

また、落語家に関連する情報「全て」ではなく、「落語家の芸名一覧」「来歴情報」など個別に取得したい場合、図 9 左上のような SPARQL クエリの記述が必要で、各情報がどのような述語（プロパティ）でリンクされているかを把握していなければならない。

このように、LOD 利用者が LOD データセット内の目的のデータを取得するための SPARQL クエリを記述するには、LOD データセットで使用されているメタデータ項目と、それらの項目がどのような構造で記述されているか、といった「グラフ構造」の把握が必要となる。

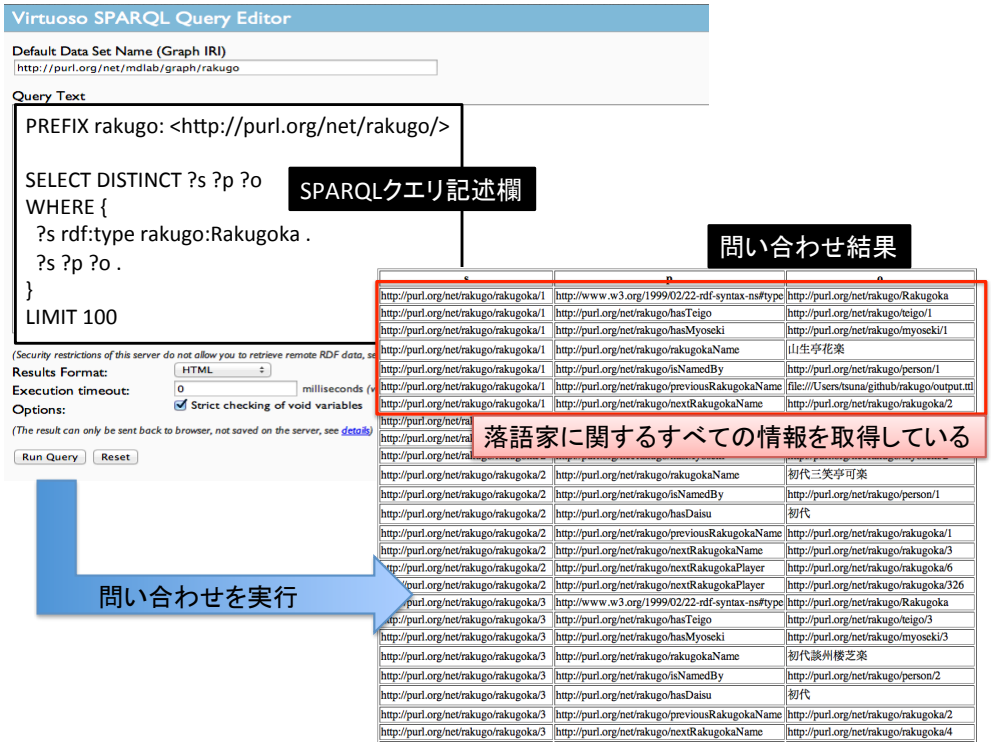


図 8 簡単な SPARQL クエリの実行

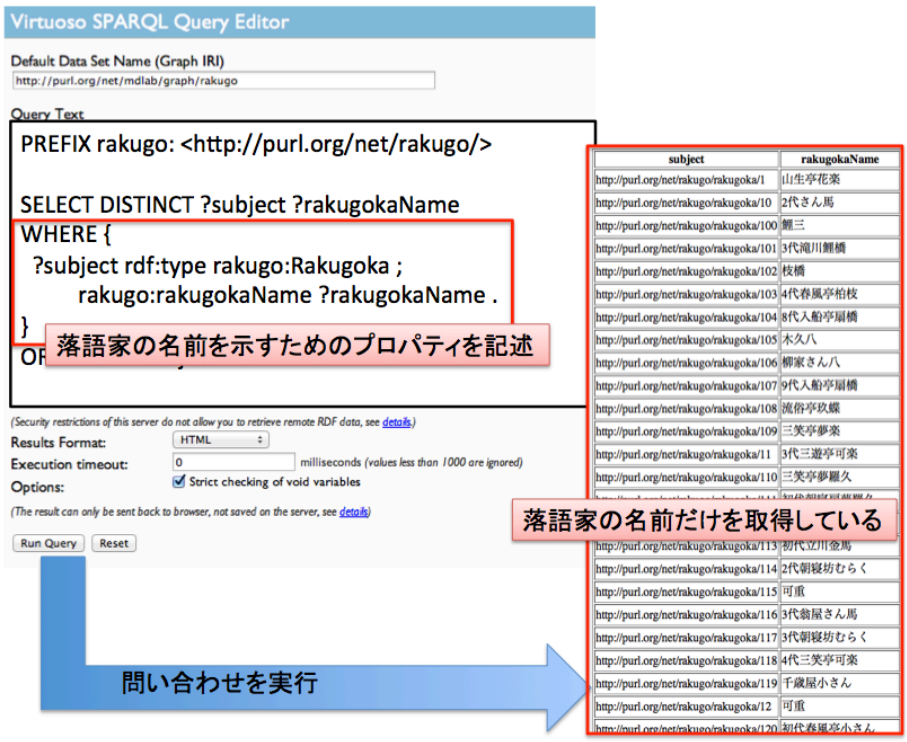


図 9 複雑な SPARQL クエリの実行

2.4. LOD の利用における問題点

2.2 節で LOD を使用した Web アプリケーションの例を挙げ、2.3 節では LOD データセットから目的のデータを取得するための構造化問い合わせ言語 SPARQL について述べた。

SPARQL クエリの記述には LOD データセットのグラフ構造に対する理解が求められるが、RDF のグラフ構造パターンは多様である。

図 10 は LOD の利用手順を表した例である。LOD 利用者は作成したい Web アプリケーションの目的などを考え、そのために必要な LOD データセットを探す。LOD データセットを利用する際、利用者が目的とするデータがそのデータセットに含まれているかを調べるためにいくつかの問い合わせを行い、グラフ構造を理解する必要がある。この作業は RDF や SPARQL に詳しい技術者でなければ大変な手間となる。この手間が LOD を利用する際の問題となっている。

このような手間を省くものが、対象となる LOD データセットのグラフ構造や、メタデータ記述規則を定義した「メタデータスキーマ」である。次章ではメタデータスキーマを用いた LOD データセットの活用について述べる。

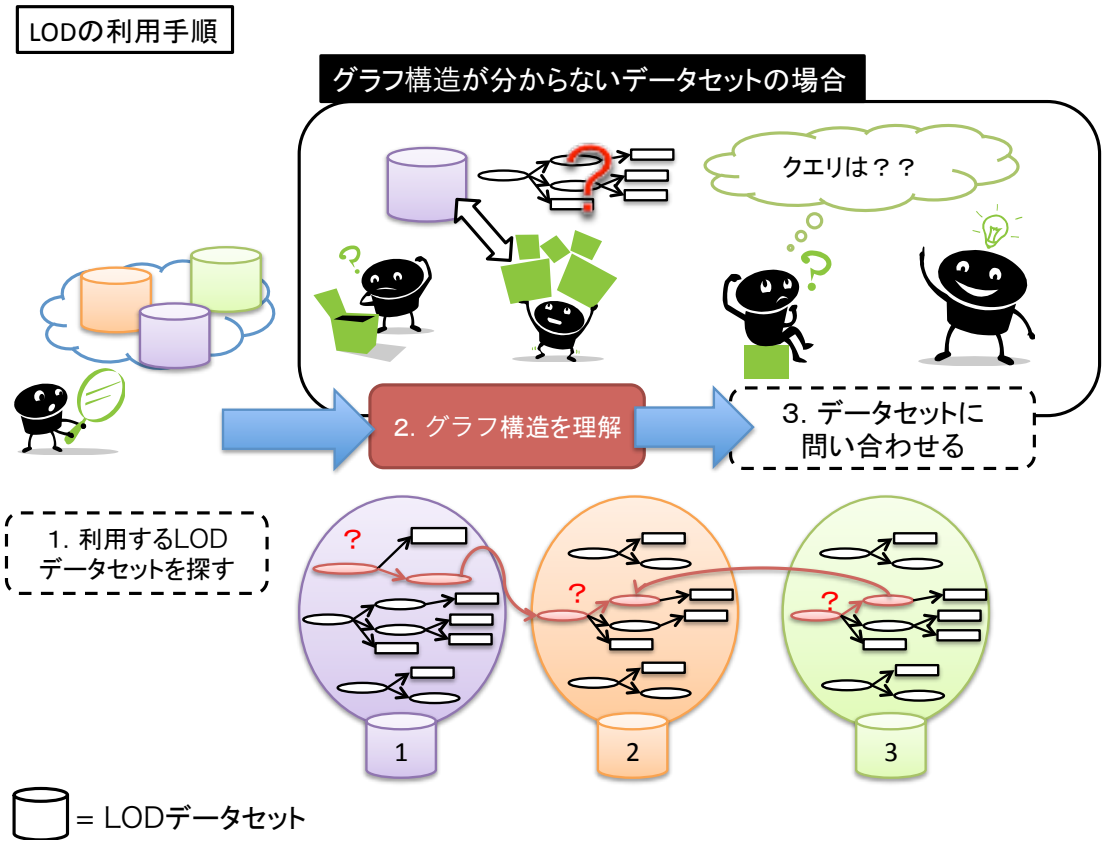


図 10 LOD の利用手順例

3. メタデータスキーマを用いたLODデータセットの活用

3章ではLODデータセットの利用について説明し、その中でのメタデータスキーマの役割を述べる。

3.1. メタデータスキーマ

メタデータスキーマは、どのような語彙を用いてどのような構造を持つメタデータを記述するのか、また、メタデータ記述項目にどのような制約条件を与えるのかといったことを定義したもののことである。メタデータスキーマを定めることで明確な規則に従ったメタデータの記述が可能となる。メタデータスキーマが明らかなLODデータセットは利用者がデータセットのグラフ構造も把握しやすくなる。

メタデータスキーマは、メタデータの記述に用いる項目と項目値を明確にするために定められる「メタデータ語彙」と、メタデータ記述の構造的制約を表す「メタデータ記述規則」から構成される。LODとして公開されるメタデータは、それぞれのデータセット公開者のみでなく、データセット利用者も作成、編集可能である。そのため、第三者によるメタデータの作成、編集によってデータセットの使用メタデータ語彙、構造が混乱しないためには、それらの定義を記述したメタデータスキーマが必要である[4]。

メタデータ語彙とはメタデータを記述し相互利用可能とするために定められた、メタデータ記述に用いる項目名(プロパティ)と対象物のタイプ(クラス)の総称である。プロパティはメタデータを記述するための「タイトル」「作成日」「記述者」などの項目のことで、RDFトリプルにおける述語部分である。クラスとはメタデータの記述対象を、書籍、人物、音楽情報などのように分類もしくは種類分けして汎化・特化関係などを用いて体系化したものである[4]。RDFでは一般的にrdf:typeプロパティによりリソースのクラスを記述する。

例えば、人に関する情報を記述するメタデータ語彙であるFOAF(Friend of A Friend)は表1から表3の様に定義されている。(引用元[25])

表 1 FOAF基本情報

名前空間（接頭辞）	foaf　※標準の接頭辞
名前空間（URI）	http://xmlns.com/foaf/0.1/
バージョン情報	0.1
タイトル	Friend of a Friend(FOAF) vocabulary
コメント	The Friend of a Friend RDF vocabulary, described using W3C RDF Schema and the Web Ontology Language.
提供者	標準提供語彙

表 2 FOAF クラス定義

ローカル名	ラベル	上位クラス	コメント
Agent	Agent		An agent(eg. person, group, ...).
Document	Document		A document.
Image	Image	foaf:Document	An image.
Person	Person	foaf:Agent	A person.

表 3 FOAF プロパティ定義

ローカル名	ラベル	上位プロパティ	定義域	値域
birthday	birthday		foaf:Agent	rdfs:Literal
depicts	depicts		foaf:Image	owl:Thing
img	image	foaf:depiction	foaf:Person	foaf:Image
made	made		foaf:Agent	owl:Thing

メタデータ記述規則とは、メタデータの記述に用いる項目の規則と、その項目は必須か任意かといった取り決めを明示的に定義したものである。例えば書誌情報に関するメタデータを記述する場合、タイトルは必須で一つのみ、著者は必須で複数可、発売日や要約は任意で一つのみ、と定めることができる。

Description Set Profile(DSP)[26]は Dublin Core Metadata Initiative(DCMI)が提唱するメタデータスキーマ定義方法である。記述形式には OWL-DSP がある[5]。OWL-DSP は DSP をオントロジー記述言語 Web Ontology Language(OWL)に基づき記述する形式である。例えばある文章の「作成者」項目の制約定義が

- プロパティとして dcterms:creator を用いる
- 入力回数の制限は「必須、一回のみ」
- 値は構造化し、氏名と年齢を記述する

の場合、OWL-DSP では次のように表現する。(Turtle 構文)

```
<#作成者> a dsp:StatementTemplate;
    rdfs:label “作成者”;
    owl:onProperty dcterms:creator;
    dsp:qualifiedCardinality 1;
    owl:onClass <#構造化作成者>;
    rdfs:comment “文書の作成者”.
```

構造化した作成者は、

- 氏名を foaf:name で必ず 1 回記述する
- 年齢を foaf:age で最大 1 回記述する

と定義したい場合は以下のように制約定義として表現できる。(Turtle 構文)

```
<#構造化作成者> a dsp:DescriptionTemplate ;
 rdfs:subClassOf [
    owl:onProperty foaf:name ;
    owl:qualifiedCardinality 1 ;
    owl:onDataRange rdfs:Literal
 ],[
    owl:onProperty foaf:age ;
    owl:maxQualifiedCardinality 1 ;
    owl:onDataRange rdfs:Literal
 ].
```

しかし、OWL-DSP による記述は専門知識がないと困難である。これに対し、平成 22 年度に行われたメタデータ情報基盤構築事業が提案している書式が「簡易 DSP」であり、表形式(TAB 区切りテキスト)の記述規則を OWL-DSP へ変換できるような書式が定められている[5]。表 4 は簡易 DSP の例で、オレンジによる編みかけ部分が OWL-DSP 記述例を表している行である。

表 4 簡易 DSP 例

[MAIN]						
#項目規則名	クラス/プロパティ	出現回数		値タイプ	値制約	説明
ID	foaf:Document	1	1	ID		
タイトル	dcterms:title	1	1	文字列		
作成者	dcterms:creator	1	-	構造化	構造化作成者	
作成日	dc:date	0	1	文字列		
[構造化作成者]						
#項目規則名	クラス/プロパティ	出現回数		値タイプ	値制約	説明
ID	foaf:Agent	1	1	ID		
名前	foaf:name	1	1	文字列		
年齢	foaf:age	0	1	文字列		

3.2. LOD 利用におけるスキーマが公開されているデータセットの不足

メタデータスキーマがあれば利用者が LOD データセットのグラフ構造を理解することができ、LOD データセットへの構造化問い合わせが容易になる。しかし、現在公開されている LOD データセットで、メタさゆりもデータスキーマを公開しているものは少ない。西出ら[3]の調査によれば、日本における Open Data 公開の取り組みである CKAN 日本語[31]、Open Data METI[32]及び Open Data のコンテストある LOD チャレンジ(2011、2012) [33]で公開されている総データセット 392 件(RDF でデータを公開しているものは 66 件)のうち、メタデータスキーマが公開されているものはわずか 4 件である。

また、メタデータスキーマの定義方法は領域、組織ごとに様々存在しており、その多くは人間が読んで理解することが前提とされている。そのため、機械によるメタデータの検証や、他組織の記述規則の再利用は難しい[4]。

メタデータスキーマが公開されている LOD データセットが不足している問題を解決するためには、メタデータスキーマが公開されていない LOD データセットのグラフ構造を分析し、メタデータスキーマを推定することが有効であると考えられる。

4. メタデータインスタンスからのスキーマ推定

4.1. LOD データセットのスキーマ推定手法の提案

本研究では、メタデータスキーマを公開していない LOD データセットが多いという問題を解決するため、メタデータインスタンスからのスキーマ推定手法を提案する。

メタデータは本来、図 11 に示したようにスキーマ設計を行いスキーマに沿って作成される。そこで本研究ではデータセットに蓄積されているメタデータインスタンスを分析し、メタデータスキーマを推定する。LOD データセットのメタデータスキーマを推定するために必要な情報と、その取得・推定方法について 4.2 節と 4.3 節で述べる。

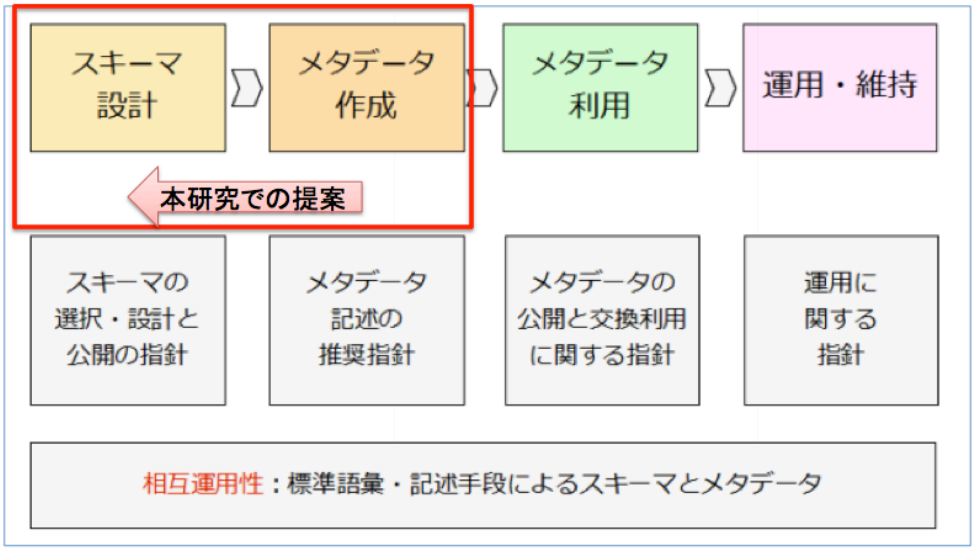


図 11 メタデータ作成手順と本研究での提案（図の引用文献[4]）

本研究ではメタデータスキーマのフォーマットとして簡易 DSP を採用し、推定するスキーマの構成要素をメタデータ情報共有のためのガイドライン[36]を参考に以下のように定めた。

- 名前空間宣言
  - 接頭辞と名前空間 URI
- 項目記述規則
  - 項目規則名(ラベル)
  - プロパティの修飾名(QName)
  - 出現回数制限(最大と最小)
  - 値タイプ
  - 値制約
  - コメント

値タイプは項目の値域を分類したもので、値が文字列なら「文字列(リテラル)」、入れ子記述に相当するリソースなら「構造化」、URI による外部参照なら「参照値」、指定がなければ「制約なし」と記述する。それぞれの値タイプに対して値制約を記述することで項目の値域の記述規則を定義するための構成要素である。例えばある項目(プロパティ)に対して値タイプを「文字列」値制約を XML スキーマデータ型「xsd:decimal」とすることでその項目は十進数を値として持つと定義することができる。

4.2. メタデータインスタンスから得られるスキーマ情報

4.2 節ではスキーマの構成要素のうち、メタデータインスタンスから得られる情報について述べる。

4.2.1. 使用タームと使用メタデータ語彙リスト

まず、LOD データセットで使用されているターム（クラス、プロパティ）を把握するため、対象 LOD データセット内で使用されているタームを取得する。

LOD データセット内に蓄積されているメタデータインスタンスには「メタデータ記述対象を表す URI」「クラスを表す URI」「プロパティを表す URI」「外部参照(外部サイトやドキュメントファイル)を表す URI」「リテラル」が含まれる(表 5 参照)。そのうち、「クラスを表す URI」と「プロパティを表す URI」を取得するためのグラフパターンを取得するための SPARQL クエリが図 12、このクエリが表しているグラフパターンを図示したものが図 13 である。

表 5 LOD データセットのリソース例とその種類

データセットのリソース例	種類
http://example.com/person/1	メタデータ記述対象
http://example.com/Person	クラス
http://example.com/name	プロパティ
http://www.tsukuba.ac.jp	外部参照
“筑波大学”	リテラル

```
# クラスを表す URI 取得用クエリ
SELECT DISTINCT ?class
WHERE {
  ?s rdf:type ?class.
}

# プロパティを表す URI 取得用クエリ
SELECT DISTINCT ?property
WHERE {
  ?s ?property ?o.
}
```

図 12 クラス、プロパティを表す URI 取得クエリ

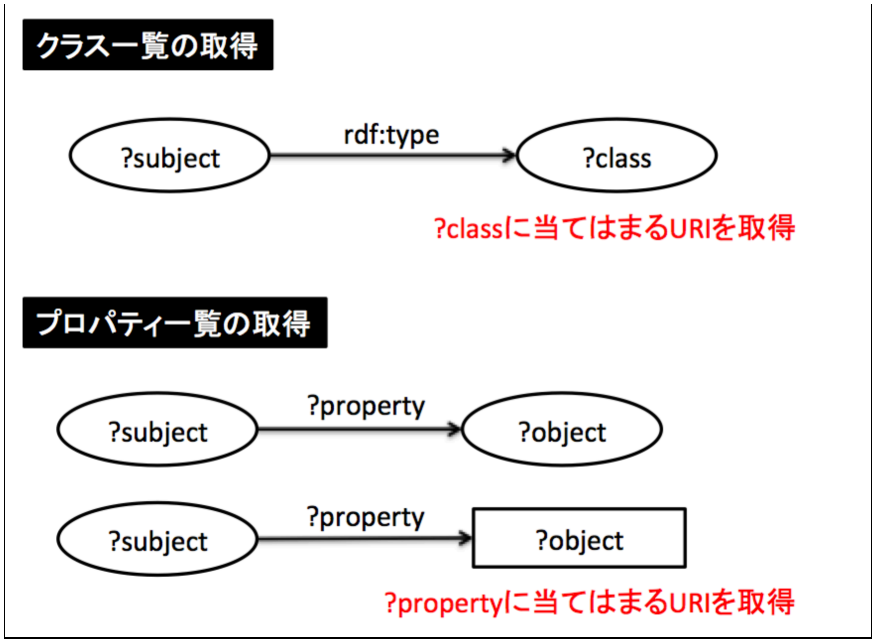


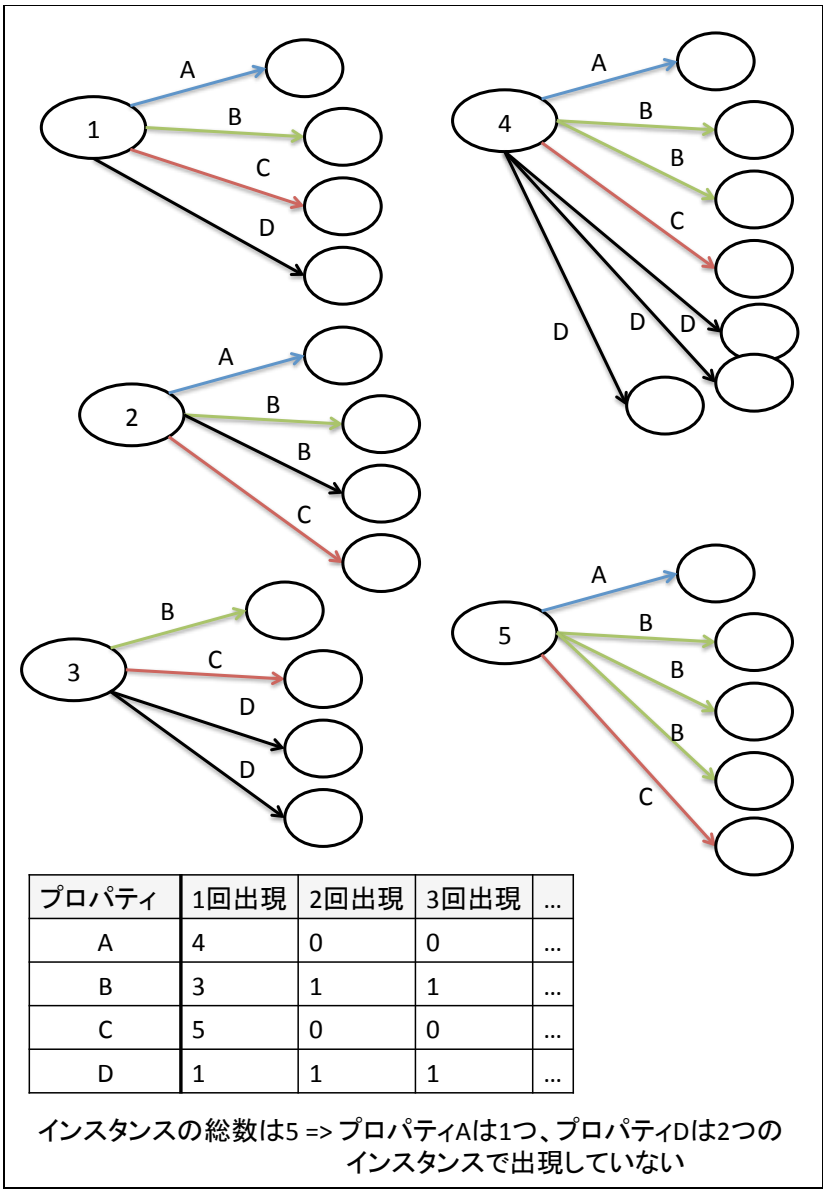
図 13 使用クラス、プロパティのグラフパターン

4.2.2. 使用プロパティの出現回数制限

スキーマの出現回数制限を定義するため、対象 LOD データセット内で使用されているプロパティの出現回数を取得する。

データセット公開者はメタデータの記述に用いる項目を定めた後、その項目は必須か任意かといった取り決めを明示的に定義する。(3.1 節参照)

簡易 DSP ではプロパティの出現回数は「最小回数」と「最大回数」の組み合わせにより「必須」なのか「推奨」なのか、何回まで記述してよいかを表現する。(表 6 参照) そのため、本研究では rdf:type をもつ全リソース(rdf:type で記述されたクラスのインスタンス)に対し、そのインスタンスを主語とするトリプルのプロパティの出現回数をカウントし、出現回数制限を抽出する。(図 14 参照) 図 15 は各プロパティの出現回数を数えるために用いる SPARQL クエリである。但し、図 15 の SPARQL クエリではプロパティが「記述されていない」インスタンス(図 14 内、プロパティ「D」が記述されていないインスタンス「5」)の個数を数えられないので、対象クラスのインスタンス数とプロパティの総出現回数の差から取得する必要がある。



プロパティ	1回出現	2回出現	3回出現	…
A	4	0	0	…
B	3	1	1	…
C	5	0	0	…
D	1	1	1	…

インスタンスの総数は5 => プロパティAは1つ、プロパティDは2つの
インスタンスで出現していない

図 14 同一クラスのインスタンスからの
プロパティ出現パターン例と出現回数の取得手順

```
# instance_class : 項目出現回数を取得したいクラス
SELECT ?s ?property (COUNT(?property) AS ?propertyCount)
WHERE {
   ?s ?property ?o.
   ?s rdf:type <instance_class>.
}
GROUP BY ?s ?property
```

図 15 使用プロパティの出現回数カウント用クエリ

表 6 簡易 DSP の出現回数制限記述例

最小出現回数	最大出現回数	意味	図 14 中の記号
0	1	記述は任意で、最大 1 回	A
1	-	必須で、何回記述してもよい	B
1	1	必須で、必ず 1 回	C
0	-	制約なし（任意で、何回出現してもよい）	D
推奨	-	推奨（検証上は任意と同意）	-

4.2.3. 各プロパティの値域

各プロパティの値タイプ・値制約を推定するため、対象 LOD データセット内で使用されているプロパティが示す値のクラス(値域)を取得する。

簡易 DSP では値域に関して「値タイプ」「値制約」で記述する。これらを取得する際のフローチャートを図 17 に示す。また、フローチャートの最初にある処理「目的語インスタンス」を取得する SPARQL クエリが図 16 である。フローチャート中の①～⑦それぞれのグラフパターンに一致する RDF グラフの例を図 18 から図 21 までに示す。

```
# instance_class : 項目出現回数を取得したいクラス
SELECT DISTINCT ?subject ?property ?object ?objProperty ?objObject ?objType
WHERE {
?subject rdf:type <instance_class>.
?subject ?property ?object.
OPTIONAL { ?object ?objProperty ?objObject.}
OPTIONAL { ?object rdf:type ?objType.}
}
```

図 16 プロパティの値域を取得するクエリ

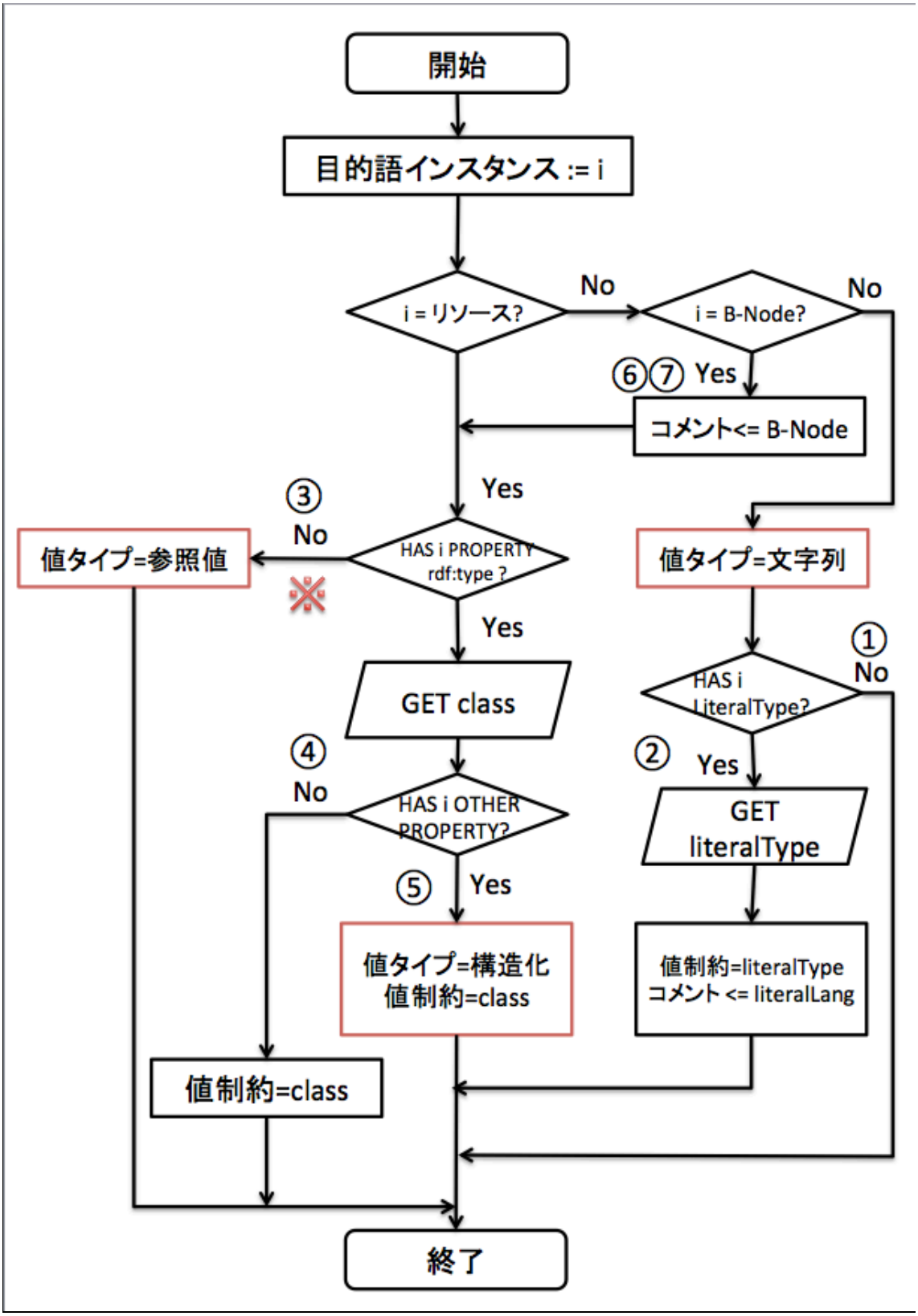


図 17 取得した目的語のインスタンス(URI)から値域を推定するフローチャート

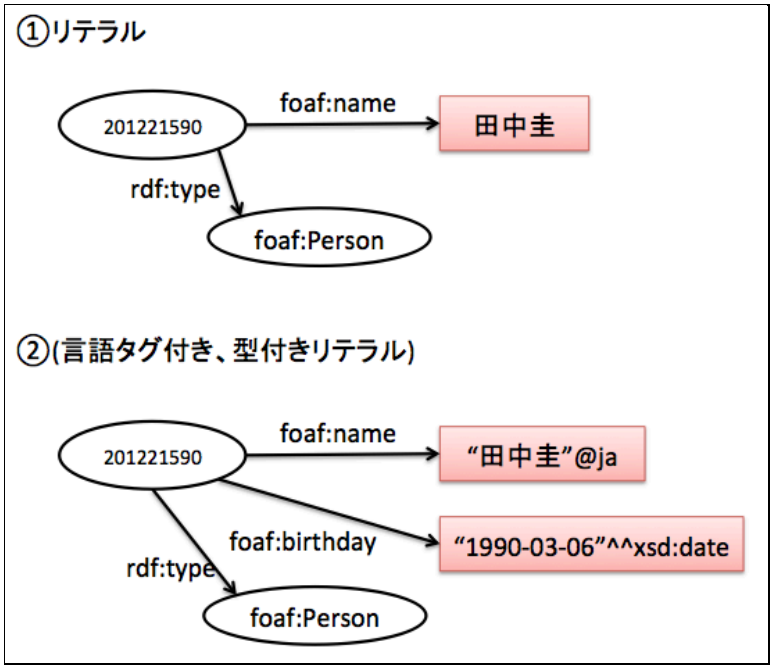


図 18 RDF グラフ例：グラフパターン①, ②

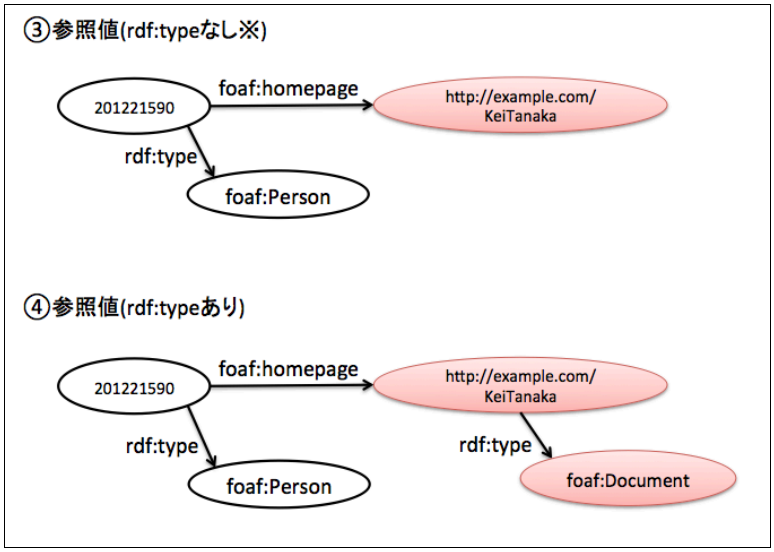


図 19 RDF グラフ例：グラフパターン③, ④

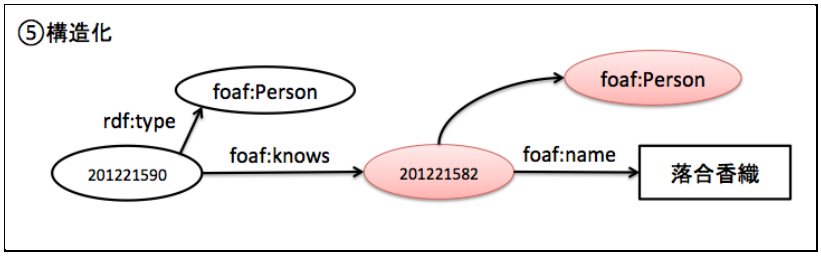


図 20 RDF グラフ例：グラフパターン⑤

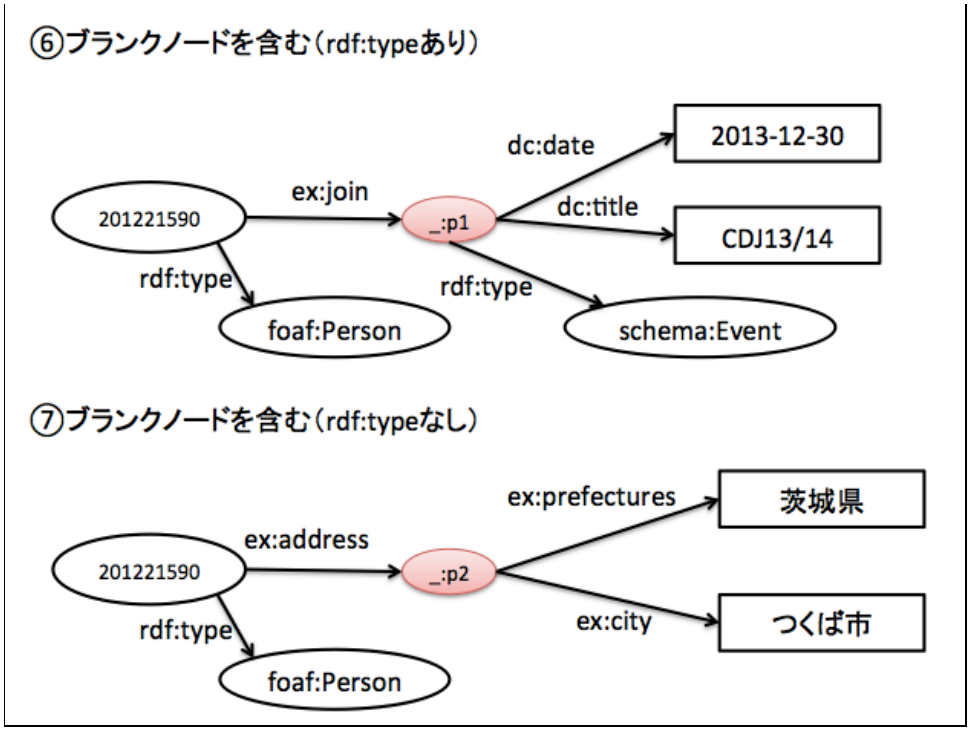


図 21 RDF グラフ例：グラフパターン⑥,⑦

パターン③において、LOD データセットで公開されている RDF のインスタンスにはクラスを暗黙的に示しているものもある。(フローチャート内の※部分で推論が必要となる)そのため、対象のインスタンスが、記述対象の識別子としての URI なのか、参照値 URI なのかを判断するためには、Web オントロジー及びプロパティの語彙定義から推定する必要がある。この時の手順は 4.2.3 節で述べる。

値域が文字列（リテラル）の場合(パターン①②)、文字列そのものであるプレーン・リテラル(plain literal)なのか、「文字の列」「整数」「日付」などの特定タイプの値として解釈できる型付リテラル(typed literal)なのかといった情報を取得する必要がある。(表 7 参照)

表 7 リテラルの種類

リテラル(文字列) 例	名称
“田中圭”	プレーン・リテラル
“田中圭”@ja	言語タグを持つプレーン・リテラル
“1990-03-06”^^xsd:date	型付リテラル

RDF では型付リテラルにつけるデータ型として主に XML スキーマ第 2 部[27]で定義されている基本データのうちから表 8 に示しているものを用いている。

表 8 型付リテラルにつける XML スキーマデータ型の一例

XML スキーマデータ型	例
文字列データ	xsd:string, xsd:language, xsd:normalizedString …
真偽値	xsd:boolean
数値データ	xsd:decimal, xsd:float, xsd:integer, xsd:byte …
日時データ	xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth …
その他	xsd:hexBinary, xsd:base64Binary, xsd:anyURI

4.2.4. メタデータ構造

取得したプロパティの値域を基に、対象データセットのメタデータ構造(グラフ構造)を推定する。LOD データセットのメタデータ構造には図 22 と図 23 で示す 4 パターンが考えられる。

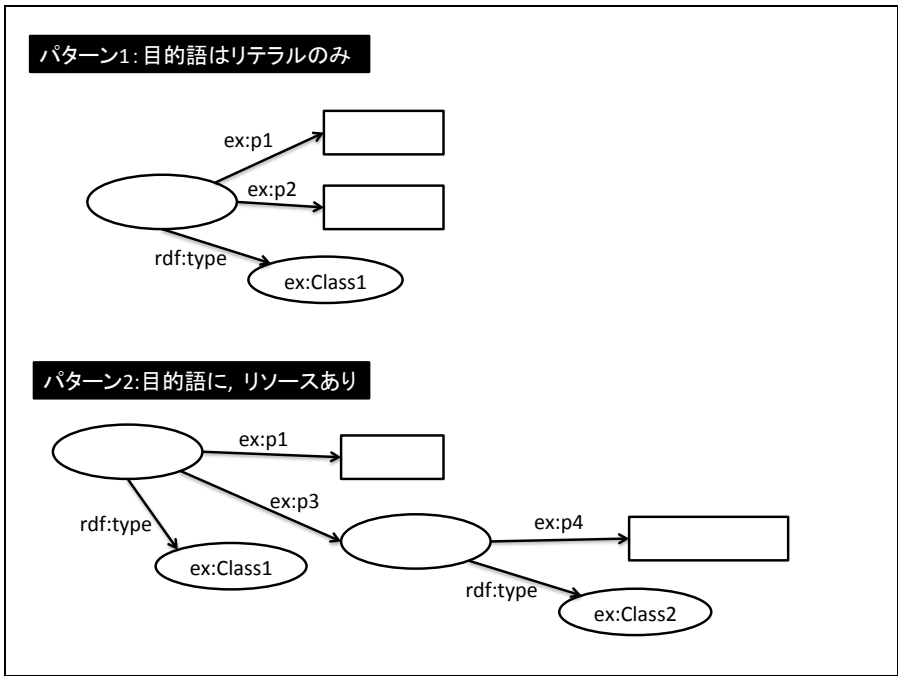


図 22 メタデータ構造 パターン 1,2

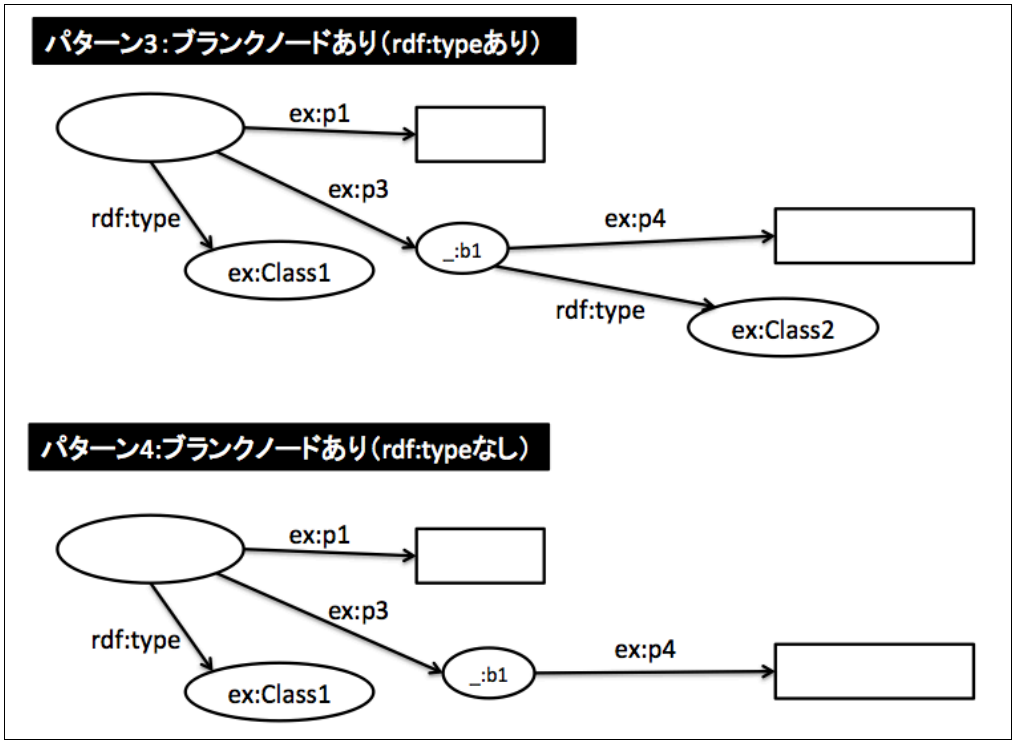


図 23 メタデータ構造 パターン 3,4

パターン 1 は目的語がリテラルのみの RDF グラフが蓄積されているパターンで、パターン 2～4 は目的語にリソースがある RDF グラフが蓄積されているパターンである。値タイプはパターン 1 が文字列、パターン 2～4 は構造化として推定される。

RDF のリソースは URI 参照で名前付けしていないブランクノードで記述することもでき、ブランクードには rdf:type によるクラスの記述があるものとないものがある。パターン 4 の様にブランクノードに rdf:type プロパティが記述されていない場合、プロパティからのクラスの推定が必要となる。

4.2.5. レコードの区切り

LOD データセットを公開する一般的な形式の一つである SPARQL Endpoint では、 RDF グラフがトリプル単位で蓄積され、蓄積前の RDF ファイルとは違うレコードの区切りとなる。レコードの区切りを元の通りに推定することは難しいため、本研究ではインスタンスがクラスを持つかどうかから 1 レコードの区切りを推定する手法を提案する。

RDF グラフパターンそれぞれのレコードの区切り方を示したものが図 24 から図 26 である。色付き線の囲いが一つのレコードである。

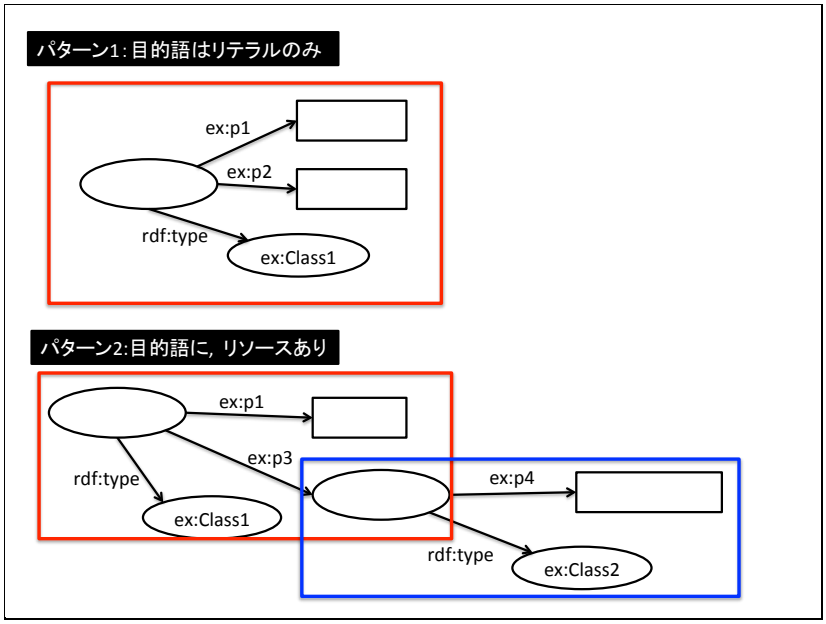


図 24 レコードの区切り：パターン 1, 2

図 24 の様に、ブランクノードが使用されていないグラフ構造であれば、クラスをもつインスタンスを基準にしてレコードの区切りを判定可能である。

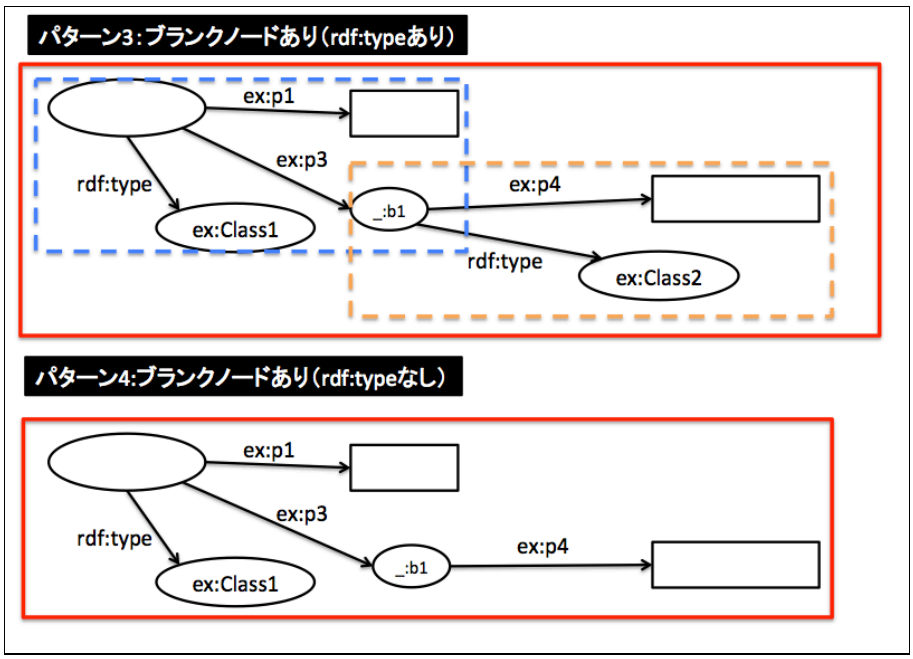


図 25 レコードの区切り：パターン 3, 4

図 25 のように、ブランクノードがある場合はブランクノードをレコードの区切りとせず、次のトリプルまでを 1 レコードとする。

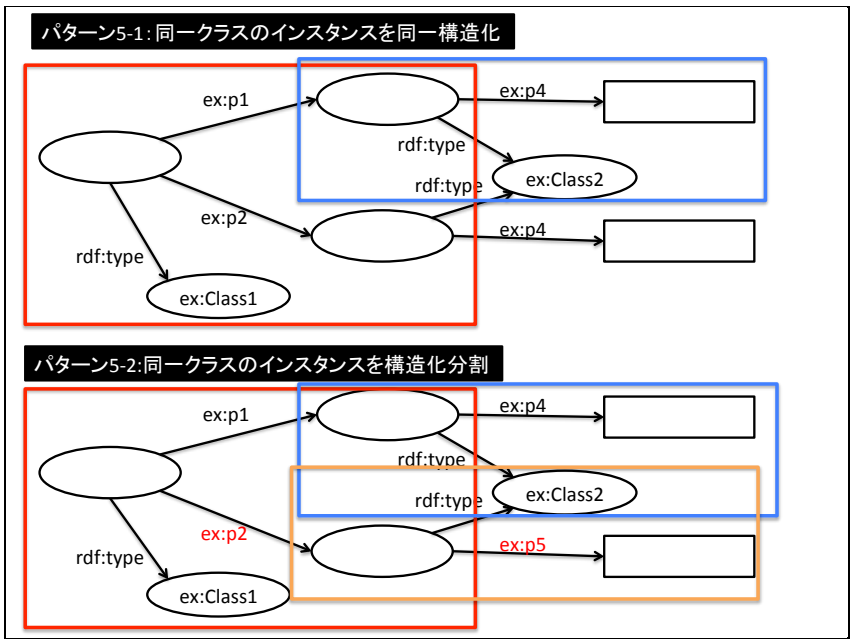


図 26 レコードの区切り：パターン 5

パターン 2 と 3 の場合に、一つのインスタンスから同一クラスのインスタンスが複数ある場合、パターン 5-2(図 26 参照)のように、同一クラスのインスタンスが異なる構造として使用されていないか確認する必要がある。例えば書籍を記述している RDF グラフで、一つの書籍から著者、編者、翻訳者全てを foaf:Person クラスで記述している場合がある。この場合、同一クラスのインスタンスに記述されているプロパティから、各インスタンスが同一用途かどうかを判別できることが望ましい。

4.3. 既存メタデータスキーマから得られるスキーマ情報

LOD ではデータセット内にリソースのクラスを直接記述せず、使用するタームのメタデータ語彙定義や、Web オントロジーによる推論を行うことでデータを表現することが可能である。そのためスキーマ推定に必要な情報の中には、公開されているメタデータ語彙定義や Web オントロジーなどの既存メタデータスキーマを用いてメタデータインスタンスのクラスや値域を推論しなければならない場合がある。これに対し本研究では、Linked Open Vocabularies[28]や Meta Bridge[29]のような、既存のメタデータスキーマやメタデータ語彙定義を蓄積・管理・公開しているサービスを用いてスキーマ情報を補完する。以下では、Linked Open Vocabularies と Meta Bridge について述べる。

Linked Open Vocabularies

Linked Open Vocabularies(以下 LOV)は、メタデータ語彙定義の蓄積と発見、共有を目的としているプロジェクトである。図 27 左上のように、メタデータ語彙同士の関連も蓄積されている。また、それぞれのメタデータ語彙に関する統計情報も蓄積されている。本研究では蓄積されているメタデータ語彙の代表的な接頭辞や定義ファイルの URI、関連するメタデータ語彙などが格納されている LOV Endpoint と、それらのメタデータ語彙全ての定義内容を一つにまとめている LOV Aggregator Endpoint を利用する。

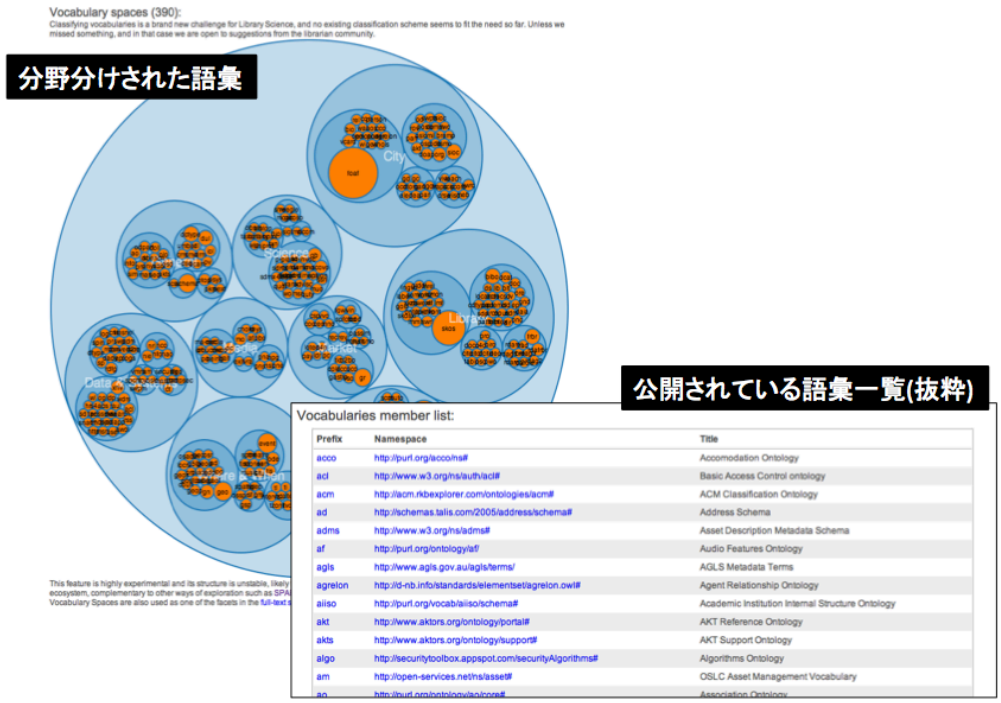


図 27 Linked Open Vocabularies

Meta Bridge

Meta Bridge は、平成 22 年度に総務省の新 ICT 利活用サービス創出支援事業として採択されたメタデータ情報基盤整備事業が開発したメタデータ情報基盤システムである。Meta Bridge は組織を限定せずに様々なコミュニティが作成したメタデータ記述項目に関するメタデータ語彙定義とメタデータ記述規則を蓄積している。蓄積する定義情報の記述形式には簡易 DSP を採用している。Meta Bridge ではユーザー登録を行うことで誰でもメタデータ語彙、メタデータ記述規則を登録することが可能で、登録したメタデータスキーマの再利用が容易である。(図 28,29 参照)

図 28 Meta Bridge 利用画面：メタデータ語彙定義一覧

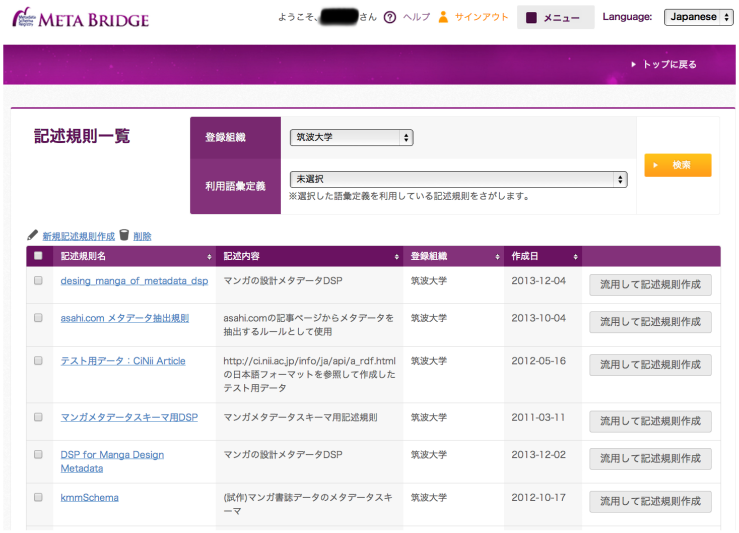


図 29 Meta Bridge 利用画面：メタデータ記述規則一覧

4.3.1. メタデータ語彙の接頭辞と名前空間 URI

LOD データセット内の RDF データを記述する際にリソースは URI で記述される。URI の記述は基本的に修飾名(QName)で行われる。例えば<http://example.com/Person>というリソースを記述する際には名前空間 URI「http://example.com/」を「ex:」という接頭辞で記述するよう宣言しておき「ex:Person」と記述する。(図 30 参照)

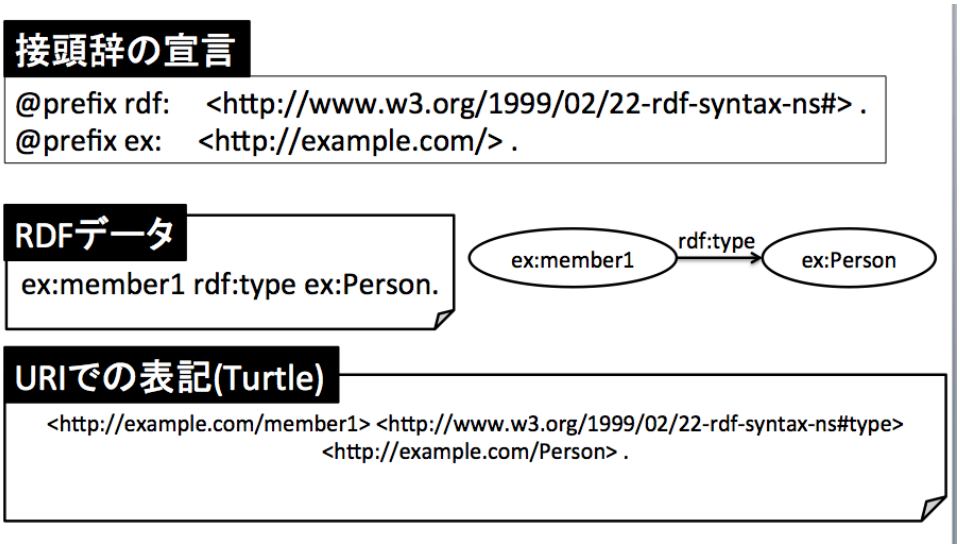


図 30 接頭辞を用いたリソース URI の記述

名前空間 URI と対応させる接頭辞はメタデータ記述者が任意に設定できるが、標準的な語彙であれば共通の接頭辞で記述した方が可読性は向上する。本研究では既存のメタデータスキーマ内で記述されている接頭辞と名前空間 URI の組を取得し、スキーマ推定対象の LOD データセット内で用いられているリソースの名前空間 URI に対して用いられている接頭辞を推定する。

4.3.2. 使用タームの語彙定義

メタデータを記述する際、メタデータの相互運用性の観点からスキーマ設計者は既存のメタデータ語彙をなるべく用いることがよいとされる。メタデータ語彙定義では各タームがプロパティなのかクラスなのか、プロパティであれば定義域と値域はなにか、その項目に関する説明が定義される。RDF グラフではインスタンスのクラスを rdf:type プロパティを用いて記述するが、プロパティの定義域、値域などからクラスを推論できるものの記述を行っていない場合もある。そのため、スキーマを推定するためには LOD データセット内で使用されているメタデータ語彙の語彙定義が必要である。

そこで、メタデータスキーマレジストリからメタデータ語彙定義を取得しスキーマ情報を補完する。今回はメタデータ語彙定義の問い合わせ先として、LOV が提供している LOV Aggregator

を使用した。使用クラスのラベル、コメント、上位クラスと使用プロパティのラベル、定義域、値域、上位プロパティがスキーマ情報と関わりがあり、これをLOV AggregatorのSPARQL Endpointから取得するクエリが図31である。

```
# instance_class : 語彙定義を取得したいクラス
# OPTIONAL : 一致するグラフパターンがある場合のみ取得する SPARQL 構文
SELECT DISTINCT *
        WHERE{
        <instance_class> rdfs:label ?label.
        OPTIONAL {<instance_class> rdfs:comment ?comment.}
        OPTIONAL {<instance_class > rdfs:isDefinedBy ?isDefinedBy.}
        OPTIONAL {<instance_class > rdfs:subClassOf ?superClass.}
}

# instance_property : 語彙定義を取得したいプロパティ
SELECT DISTINCT *
WHERE{
    <instance_property> rdfs:label ?label.
    OPTIONAL {<instance_property> rdfs:comment ?comment.}
    OPTIONAL {<instance_property> rdfs:isDefinedBy ?isDefinedBy.}
    OPTIONAL {<instance_property> rdfs:domain ?domain.}
    OPTIONAL {<instance_property> rdfs:range ?range.}
    OPTIONAL{<instance_property> rdfs:subPropertyOf ?superProperties.}
}
```

図 31 各クラス、プロパティのメタデータ語彙定義取得クエリ

4.3.3. データセット内で記述されていないプロパティの値域(目的語のクラス)

4.2.3節で述べたように、LODデータセットのRDFデータには必ずしもrdf:typeプロパティによってインスタンスのクラスが直接記述されていない。そのため、4.3.2節で取得したメタデータ語彙定義内のプロパティ定義情報から値域(値制約)を補完する必要がある。

例えばとあるRDFデータセットにはdcterms:languageプロパティで記述されている値(インスタンス)にクラスの記述がない。そこでDC Termsのメタデータ語彙定義を取得すると、dcterms:languageは値域にdcterms:LinguisticSystemが定義されている(表9参照)。よってLOV Endpointのスキーマ情報としてdcterms:languageの値域(値制約)はdcterms:LinguisticSystemクラスとなる。

表 9 メタデータ語彙定義から取得したあるデータセットのプロパティ定義内容（抜粋）

PROPERTY	LABEL	DOMAIN	RANGE
foaf:name	Complete Name		
dcterms:language	Language	_	dcterms:LinguisticSystem
dcterms:issued	dct:issued		rdfs:Literal
foaf:homepage	homepage	owl:Thing	foaf:Document
lvont:representedBy	represented by	_	_
void:dataDump	Data Dump	void:Dataset	rdfs:Resource
vs:term_status	term status	_	_
cc:license	licence	cc:Work	cc:License
dcterms:date	Date	_	rdfs:Literal
foaf:account	account	foaf:Agent	foaf:OnlineAccount

4.4. メタデータスキーマ推定手順

4.2 節、4.3 節で述べたスキーマ情報を取得しスキーマ推定を行うアルゴリズムを図 32 に示す。手順を示すものであり、各関数の処理内容は 5 章で述べるものとする。

```
begin endpoint = endpoint_uri
  #  対象データセットで使用されているメタデータ語彙 (4.2.1)
  extracted_terms_class = get_all_classes(endpoint_uri);
  extracted_terms_property = get_all_properties(endpoint_uri);

  # 接頭辞、名前空間の組み合わせ (4.3.1)
  namespace_uris_and_prefix = get_prefix(extracted_terms_class, extracted_terms_property)

  # 使用されているタームのメタデータ語彙定義 (4.3.2)
  terms_definitions = get_definition(extracted_terms_class, extracted_terms_property)

  repeat extracted_terms_class := klass
     # プロパティの出現回数制限 (4.2.2)
     occurrence_of_instance_per_class = count_instance_per_class(endpoint_uri, klass)
     occurrence_of_property_per_class = count_property_per_class(endpoint_uri, klass,
                                                    occurrence_of_instance_per_class)
     # プロパティの値域取得 (4.2.3, 4.3.3)
     extracted_object_type = extract_object_type_per_class(endpoint_uri, klass, terms_definitions)

  until extracted_terms_class.nil

  # メタデータ構造、レコード区切り (4.2.4, 4.2.5)
  description_set_profile = get_dsp(occurrence_of_instance_per_class,
                                    occurrence_of_property_per_class, extracted_object_type,
                                    namespace_uris_and_prefix)
end
```

図 32 スキーマ推定のアルゴリズム

5. メタデータスキーマ推定システムの実現

メタデータインスタンスからのスキーマ推定手法の実現として、プログラミング言語 Ruby を用いてメタデータスキーマ推定システムを開発した。RDF データの処理にはライブラリ「RDF.rb」を用い、SPARQL Endpoint への問い合わせにはライブラリ「SPARQL」[43]「SPARQL Client」[44]を用いた。

5.1. システムの概要

図 33 は作成したスキーマ推定システム(以下 本システム)の構成を表した図である。

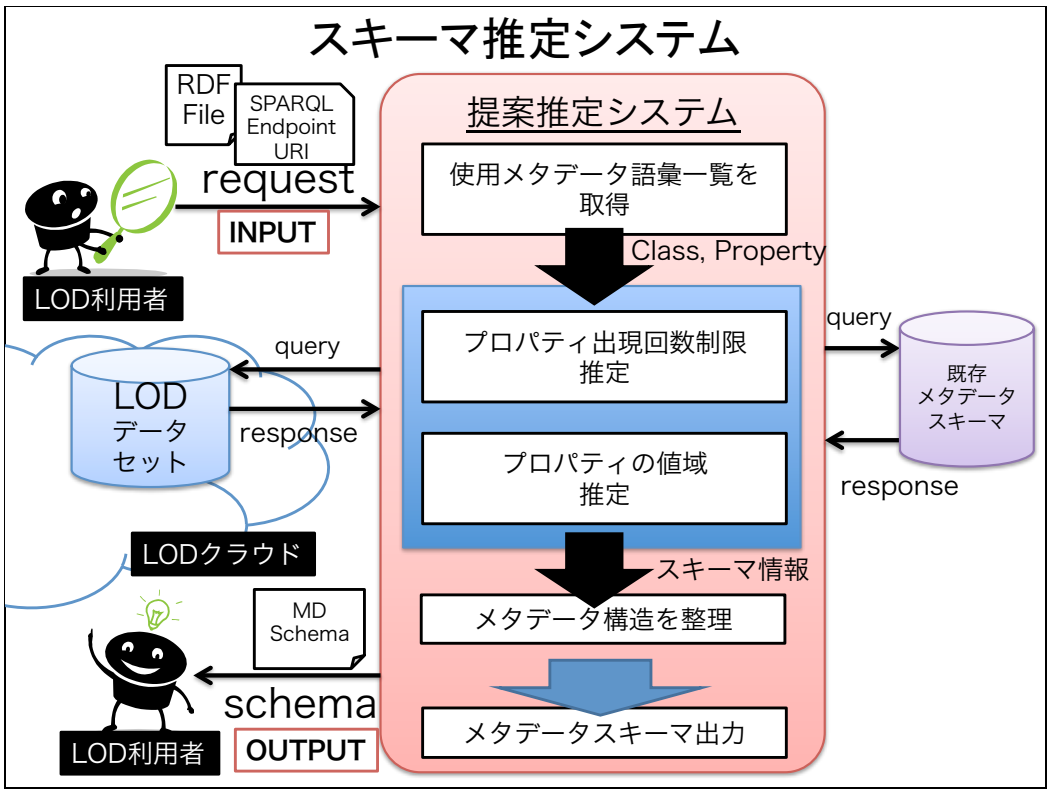


図 33 スキーマ推定システムの概要図

本システムはコマンドラインインタフェースを通じて利用する。本システムに実装されている入力コマンドと、入力値の意味を指定するためのオプションは以下の通りである。

```
Commands:
   bin/extract execute [-egio]  # スキーマ推定機能 (5.2.1～5.2.3)
   bin/extract graph [-e]       # Graph URI 取得機能 (5.2.4)
   bin/extract suggest [-eg]    # MAIN ID 提案機能 (5.2.4)
Options:
  –e[--endpoint]  #  SPARQL Endpoint URI または RDF ファイル名(必須)
  -g[--graph]     # Graph URI (オプション)
  -i[--id]        # MAIN I URI (オプション)
  -o[--output]    # 出力される簡易 DSP のファイル名 (オプション)
```

以上のコマンドを実行すると、本システムが LOD データセット及び既存スキーマを蓄積しているメタデータスキーマレジストリへの問い合わせを行い、「簡易 DSP(TSV または CSV 形式)」と「各機能で取得した情報の CSV ファイル」を出力する。図 33 内の各機能について 5.2 節で述べる。

5.2. 推定システムの機能

5.2.1. 使用メタデータ語彙一覧取得

データセット内で使用されているメタデータ項目とその語彙定義を取得する機能。取得手法は 4.2.1 節,4.3.1 節,4.3.2 節で述べている。接頭辞の候補が取得できなかった場合は接頭辞を「ns1,ns2,…」として出力する。

入力:「SPARQL Endpoint URI または RDF ファイル名」「Graph URI」
出力:「使用プロパティ・クラス」「プロパティ・クラス URI の名前空間 URI を表す接頭辞一覧」「プロパティ・クラスのメタデータ語彙定義」

本システムは以下のようなコマンドで推定を開始し、5.2.1 節から 5.2.3 節までの機能が実行される。

コマンド例

bin/extract execute -e “http://mdlab.slis.tsukuba.ac.jp/sparql” -g “http://purl.org/net/mdlab/graph/rakugo”

5.2.2. タームの記述規則推定

各クラスのインスタンスに記述されている項目の記述規則を推定する。この機能は 5.2.1 節で実行したコマンドから連続で実行され、対象データセットで使用されているクラスの数だけ繰り返される。

5.2.2.1. プロパティ出現回数制限推定

(入力):「SPARQL Endpoint URI または RDF ファイル名」「Graph URI(オプション)」「推定するクラスの URI」
出力:「各クラスの総インスタンス数」「各クラスのインスタンスに記述されるプロパティの出現回数カウント」

プロパティの出現回数は 0 回から 9 回までとそれ以上の 11 段階でカウントする。

5.2.2.2. プロパティの値域推定

（入力）:「SPARQL Endpoint URI または RDF ファイル名（必須）」「Graph URI（オプション）」
「推定するクラスの URI」
出力 :「各プロパティの値のクラス、クラスごとのカウント」「コメント」

まずインスタンスから値域情報を取得し、不足している場合は既存メタデータスキーマから補完する。

簡易 DSP では一つの構造化に対して同一のプロパティに異なる値タイプ・値制約を定義することができない。そのため、本機能では値タイプ・値制約が複数取得できた場合は出現回数の多い方を採用する。また、二番目以降の候補については出現回数の全体数から判断して極めて少ないパターンの場合は排除し、そうでない場合はコメントアウトした形で DSP に出力されるようになっている。

値タイプが文字列の場合は型付リテラルかどうかを判別し、型付リテラルであればデータタイプをコメントとして出力する。

5.2.3. メタデータスキーマ出力

（入力）:「各クラスの総インスタンス数」「各クラスのインスタンスに記述されるプロパティの出現回数カウント」「各プロパティの値のクラス、クラスごとのカウント」「コメント」
出力 : 簡易 DSP（MAIN ID にする URI を記述していれば TSV, していなければ CSV）

これまで機能で取得・推定している情報からメタデータ構造を整理し、簡易 DSP の構成要素を推定し、メタデータスキーマを出力する。スキーマが推定されるとシステムは最後に以下のように生成された簡易 DSP のファイル名と、各機能で取得した情報の出力先ディレクトリ名を表示し、終了する。この時、TSV で出力した簡易 DSP はそのまま Meta Bridge に登録が可能である。

```
Simple DSP [output_01151144.csv] created!
Extracted Information about Schema => extract01151144
```

5.2.4. システム利用者の支援機能

利用者が入力値を決める支援として、本システムはスキーマ推定機能に追加して Graph URI 取得機能と MAIN ID 提案機能を実装している。

Graph URI 取得機能

SPARQL Endpoint には SPARQL Endpoint 内のトリプルをグルーピングするための識別子である Graph URI がある。SPARQL クエリで Graph URI を指定することで一つの SPARQL Endpoint に蓄積されている複数のデータセットを識別できる。複数のデータセットを公開している SPARQL Endpoint に対して Graph URI を指定せずに本システムを実行すると正しいスキーマが推定できない。そのため、実行対象とした SPARQL Endpoint に Graph URI が設定されているかを検出する必要がある。

本機能は、「SPARQL Endpoint URI」を入力値として、対象 SPARQL Endpoint 内の「Graph URI」を取得する。Graph URI が設定されていない場合は何も返さない。

MAIN ID 提案機能

MAIN ID とは簡易 DSP において先頭の ID となるクラス URI である。これは簡易 DSP の仕様により MAIN ID が定義されていなければならないため、簡易 DSP フォーマットとしてスキーマを出力し、Meta Bridge へスキーマを登録したいシステム利用者が必要とする値である。

本システムでは「SPARQL Endpoint URI」「Graph URI」を入力値として MAIN URI の候補となるクラス URI の一覧を提案する。MAIN ID は対象データセット内のメタデータ構造において先頭となるもの(自己参照を除き MAIN ID の構造化が値制限として定義されていないもの)が望ましいため、提案機能ではクラス URI がリンクされているインスタンスのうち、目的語になっていないものを優先して提案する。(図 34 参照)

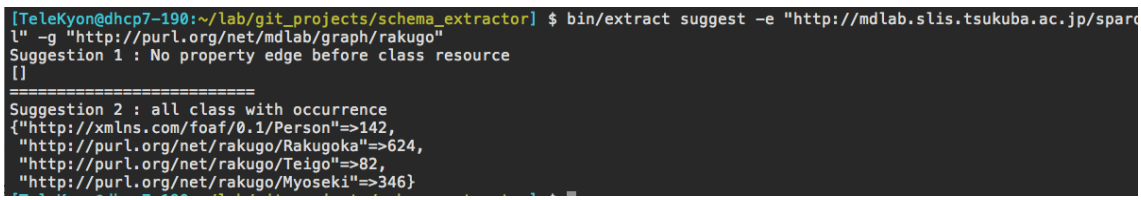


図 34 MAIN ID 提案機能　利用画面

5.3. 推定システムの利用例

落語家 LOD を利用したい LOD データセットであると仮定して本システムの利用例を示す。システム利用者が以下のようなコマンドでシステムに入力値を与えると、スキーマ推定機能が実行される。実行順に図 35, 36, 37 のように取得情報がコマンドライン上に表示される。

```
$ bin/extract execute --endpoint "http://mdlab.slis.tsukuba.ac.jp/sparql"
                --graph "http://purl.org/net/mdlab/graph/rakugo" --id "http://purl.org/net/rakugo/Rakugoka"
```

図 35 本システム実行画面 1 ：使用メタデータ語彙一覧取得

図 36 本システム実行画面２：プロパティ出現回数、値域推定

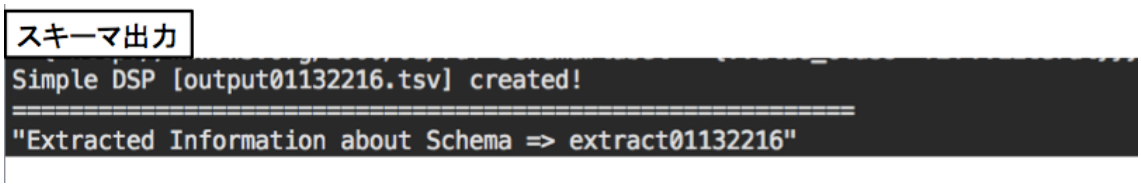


図 37 本システム実行画面３：メタデータスキーマ出力

本システムによって図 38 に示す簡易 DSP が出力される。

[@NS]						
#接頭辞	名前空間URI					
foaf	http://xmlns.com/foaf/0.1/					
ns1	http://purl.org/net/rakugo/					
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#					
rdfs	http://www.w3.org/2000/01/rdf-schema#					
[MAIN]						
#項目規則名	プロパティ	最小	最大	値タイプ	値制約	説明
ThisClassID	ns1:Rakugoka	1	1	ID		
hasTeigo	ns1:hasTeigo	0	1	構造化	#構造化ns1_Teigo	
hasMyoseki	ns1:hasMyoseki	1	1	構造化	#構造化ns1_Myoseki	
rakugokaName	ns1:rakugokaName	1	1	文字列		
isNamedBy	ns1:isNamedBy	1	1	構造化	#構造化foaf_Person	
#previousRakugokaName_ref	ns1:previousRakugokaName	1	1	参照値		maybe error value constraint. link to triple.
previousRakugokaName_con_N	ns1:previousRakugokaName	1	1	構造化	#MAIN	link to triple.
nextRakugokaName	ns1:nextRakugokaName	0	1	構造化	#MAIN	
hasDaisu	ns1:hasDaisu	0	1	文字列		
nextRakugokaPlayer	ns1:nextRakugokaPlayer	0	-	構造化	#MAIN	
previousRakugokaPlayer	ns1:previousRakugokaPlayer	0	1	構造化	#MAIN	
[構造化foaf_Person]						
#項目規則名	プロパティ	最小	最大	値タイプ	値制約	説明
ThisClassID	foaf:Person	1	1	ID		
name	foaf:name	0	1	文字列		
[構造化ns1_Teigo]						
#項目規則名	プロパティ	最小	最大	値タイプ	値制約	説明
ThisClassID	ns1:Teigo	1	1	ID		
label	rdfs:label	1	1	文字列		
[構造化ns1_Myoseki]						
#項目規則名	プロパティ	最小	最大	値タイプ	値制約	説明
ThisClassID	ns1:Myoseki	1	1	ID		
label	rdfs:label	1	1	文字列		

図 38 出力される簡易 DSP

6. 評価

6.1. LOD データセットを用いた評価実験

LOD チャレンジ Japan 2013 データセット部門にエントリーされている LOD データセットのうち、データを RDF で記述しているものを対象としてスキーマ推定の評価実験を行った。まず、LOD チャレンジ 2013 データセット部門にエントリーされているデータセットのうち、データが RDF で記述されているものを以下の条件で分類した。

A) RDF データをファイルとしてのみ公開しているもの
B) RDF データを問い合わせ可能な SPARQL Endpoint のみ公開しているもの
C) RDF データをファイル及び SPARQL Endpoint の両方で公開しているもの

分類の結果、データセット部門にエントリーされている 101 件のデータセットのうち、RDF で記述したデータを登録しているものが 51 件あった。そのうち、分類 A が 39 件、分類 B が 3 件、分類 C が 9 件あった。同一のエントリー者が複数の作品を登録しているなどを排除し、分類 A から 30 件、分類 B から 3 件、分類 C から 8 件のデータセットを実験で用いた。なお、本システムで用いている SPARQL 問い合わせライブラリに対応していない SPARQL Endpoint は分類 B,C の条件を満たさないものとした。

6.2. 実験手順

実験はそれぞれの分類に対して以下の手順で行った。なお、予備実験を行ったところ一定時間以上システムを実行するとタイムアウトとなることがわかった。そのため、本実験では問い合わせ先のサーバへの負荷も考慮し、分類に関わらずシステムを実行開始してから 1 時間を制限時間とし、1 時間経過しても推定が終了していない場合は処理を終了している。

分類 A

1. RDF ファイルを実験用 PC にダウンロード
2. 本システムを実行
3. 推定成功 or エラー箇所のメモ

分類 B

1. SPARQL Endpoint が問い合わせ可能な状態か調べる
2. 本システムを実行
3. 推定成功 or エラー箇所のメモ

分類 C （分類 A,B の実験手順をそれぞれ実行）

本実験で対象とした LOD チャレンジ 2013 データセット部門のデータセットは、rdf:type プロパティによるインスタンスクラス記述が全くないデータが見られた。このような場合、本システムではスキーマが推定できない。そこで、分類 A および分類 C の RDF ファイルへのシステムを実行しインスタンスクラスが記述されていないために不完全なスキーマが推定された場合は、RDF トリプルの主語インスタンスにテスト用クラス <http://mdlab.slis.tsukuba.ac.jp/schema/extractor/TestClass>を与えて再度実験を行った。

6.3. 評価基準

本システムは各機能でスキーマ定義情報の抽出が成功した場合は中途ファイルとして CSV 形式でその情報を出力するように実装している。この評価実験では各データセットに対して本システムを実行し、「スキーマが出力されるかどうか」「中途ファイルがいくつ出力されているか（システムの機能がどこまで実行されたか)」を集計した。中途ファイルの出力順を表 9 に示す。

表 9 スキーマ推定システムの中途ファイル出力順序

評価順序	中途ファイルの内容
1	使用プロパティ、クラス一覧
2	接頭辞と名前空間 URI の組み合わせ
3	使用プロパティ、クラスのメタデータ語彙定義
4	使用インスタンスの個数
5	使用プロパティの出現回数
6	値タイプ、値制約情報

中途ファイル出力数と、LOD データセットへの問い合わせ支援の内容の対応を表 10 に示す。

表 10 中途ファイル出力数と LOD 問い合わせ支援の対応

出力数	LOD データセットへの問い合わせ支援内容
0	(システムが実行されなかった)
3	回数の取得が制限時間以内に終了しなかった。 問い合わせに用いる項目の一覧は取得できている。
5	プロパティの値のクラス取得が実行完了していない。クラスごとのインスタンス数が取得できているので、データセット内の主な記述対象が分かる。
6	スキーマ定義項目を取得できた。データ構造まで提示できている。

6.4. 結果

実験結果を表 11 に示す。

表 11 実験結果

	中途ファイル出力数				出力スキーマ数	データセット数
	0	3	5	6		
分類 A	1	7	0	22	22	30
分類 B	1	0	0	2	2	3
分類 C(ファイル)	2	4	1	1	1	
分類 C(Endpoint)	1	2	0	5	7	8

分類 A

30 件中 22 件のデータセットでスキーマが出力された。トリプル数が著しく多いファイルは処理が終了せずに評価順序 3 まで実行された。

分類 B

3 件中 2 件でスキーマが出力された。1 件は対象の SPARQL Endpoint がシステムで利用している SPARQL クエリ構文に対応していないため中途ファイルも出力されなかった。

分類 C(ファイル)

8 件中 1 件でスキーマが出力された。RDF ファイルのトリプル数を調べたところ、推定が全て完了したものは約 2100 件であったのに対し、その他は 20000 件から 1600000 件であった。

分類 C(Endpoint)

8 件中 7 件はスキーマが出力された。ただし、インスタンスに rdf:type プロパティを記述していないデータセット 2 件では各クラスのインスタンスに対して処理されず、評価順序 3 まで実行された。

7. 考察と課題

既存メタデータスキーマを用いたメタデータインスタンスからのスキーマ推定手法と、実装したメタデータスキーマ推定システムには以下の課題が残されている

1) 参照したメタデータ語彙定義と矛盾したスキーマ設計によるデータセットもしくは項目の記述ミスがある LOD データセットが正しく推定できない
2) 統制語彙(概念や用語の関係性)や語彙定義を記述しているデータセットのスキーマ推定が不十分
3) SPARQL Endpoint の SPARQL 構文の実装内容にばらつきがあり、全ての SPARQL Endpoint に対応するにはコストがかかる
4) 大きなデータセットのスキーマ推定に時間がかかる
5) データセット公開者が推奨にしたいタームがあってもインスタンスがなければ取得できない

1) 参照したメタデータ語彙定義と矛盾したスキーマ設計によるデータセットもしくは項目の記述ミスがある LOD データセットが正しく推定できない

LOD は技術者だけではなく技術を把握していない人々もデータセットを公開している。そのため、本手法で参照したメタデータ語彙定義や既存のスキーマと異なる内容で記述されたメタデータが公開されているものがある。例えば評価実験ではテストクラスを付与して対応した rdf:type プロパティによるインスタンスクラスの記述がないもの、地理情報を記述したデータセットでは経度を表す geo:long というプロパティを geo:lng と脱字して記述しているものがあった。このような場合、本手法によるスキーマ推定を行うと LOD データセット公開者の意図と異なる推定結果が出力される可能性がある。この課題の解決として、LOD データセット公開者・データ作成者は RDF データが正しく記述されているかを容易に確認することができ、間違っている場合に正しい書式を公開者・作成者に提案する取り組みが求められる。

2) 統制語彙(概念や用語の関係性)や語彙定義を記述しているデータセットのスキーマ推定が不十分

LOD データセットには各分野の概念や用語の関係性を体系的に定義した統制語彙や、LOD データセットの記述に用いるメタデータ語彙定義を、Web オントロジーとして OWL を用いて構築し、公開しているものもある。例えば「ねじ LOD」はねじ製品を分類するねじコード(N 研コード)体系を LOD 化したものである[39]。このデータセットでは「大分類」「中分類」「小分類」「細分類」それぞれをインスタンスとして図 39 の様に記述している。ねじ LOD では、下位概念を表すインスタンスから上位概念を表すインスタンスへのプロパティに rdf:type を使用しているため本手法によるクラス(構造化)の取得を行うと 図 40 の水色枠のように rdf:type が記述されている大

分類、細分類の数だけクラスが取得され、rdf:type が記述されていない中分類、小分類を表すインスタンスは無視され、構造は取得されない。この場合、図 41 のように分類ごとの構造として分割すべきであるが、本手法では rdf:type プロパティをクラスの取得に用いているためそれが不可能である。これに対しては統制語彙、語彙定義を定義しているデータセットのための構造化分割手法が必要である。

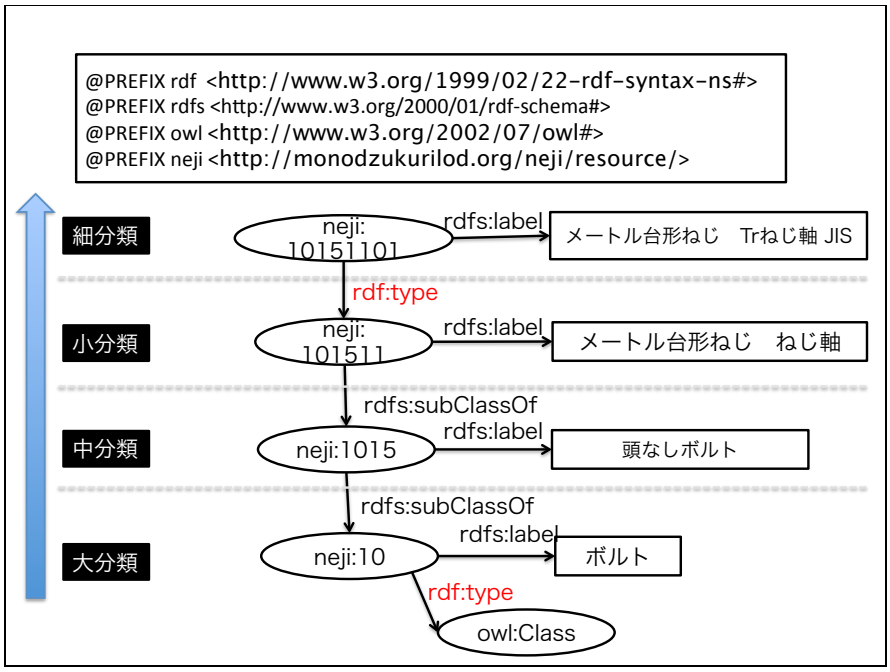


図 39 オントロジーを記述した RDF グラフの例

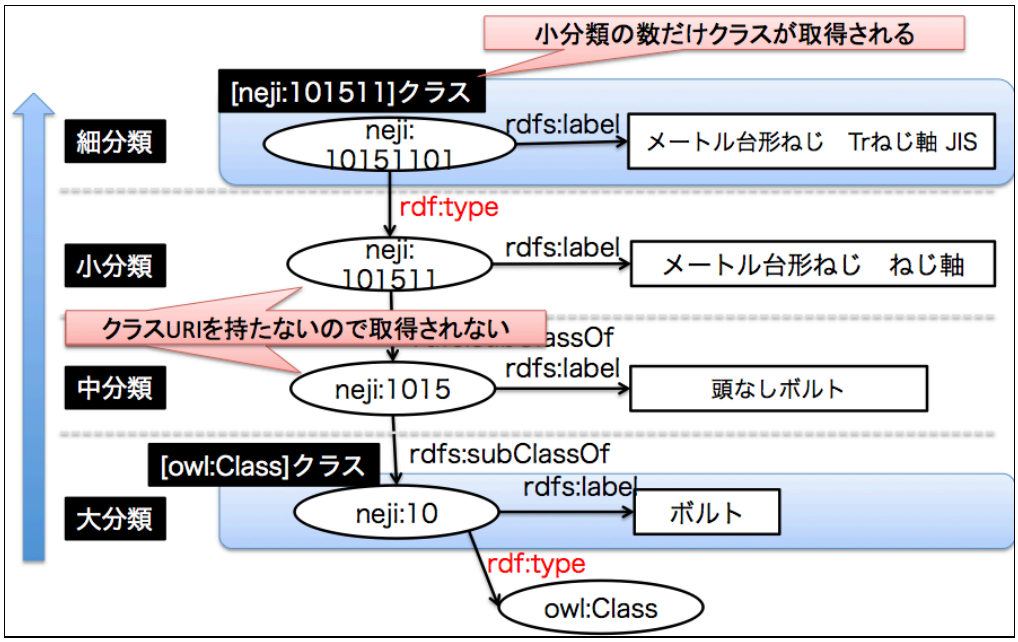


図 40 本システムで推定される構造化（青の矩形）

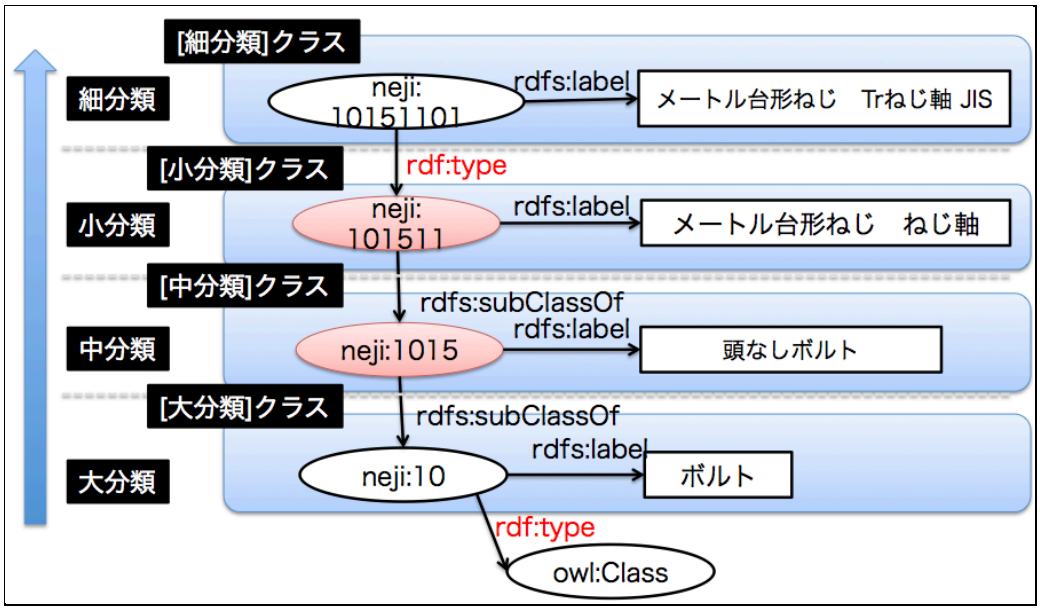


図 41 期待される構造化（青の矩形）

3) SPARQL Endpoint の実装内容にばらつきがあり、全ての SPARQL Endpoint に対応するにはコストがかかる

SPARQL の仕様では RDF に対して様々な問い合わせを行えるように多くの構文が定義されている。しかしながら、それらの構文を用いた SPARQL クエリに対応しているかどうかは SPARQL Endpoint の実装に依存する。2.3 節でも挙げたように、SPARQL Endpoint を提供するためのサービスは複数存在し、それらのサービスで全ての SPARQL の構文が実装されているとはいえず、サービスや公開者の利用しているサーバによって利用できる構文は統一されていない。実験対象にした SPARQL Endpoint の中にも問い合わせ結果数を出力する COUNT 構文が利用できないために出現回数制限の推定でエラーがでているもの、Graph URI を指定するための FROM 構文が利用できないためシステムが実行されないものがあった。この解決には対象の SPARQL Endpoint の SPARQL 構文実装状況を機械的に取得するような機能が必要である。そうして図 42 のように推定対象の SPARQL Endpoint の実装内容に合わせてシステムが推定するために問い合わせる SPARQL クエリを変更することでスキーマ推定が可能となる。しかし、現状では SPARQL Endpoint の実装状況はテストクエリを用いてヒトが調べている。推定システムを実行するたびに調査を行うことは非常にコストがかかる。2013 年 3 月に W3C で勧告された、SPARQL Endpoint についての情報を記述する「SPARQL 1.1 Service Description」[37]によって SPARQL Endpoint に関する情報が SPARQL 問い合わせで取得できるようになることが望まれる。

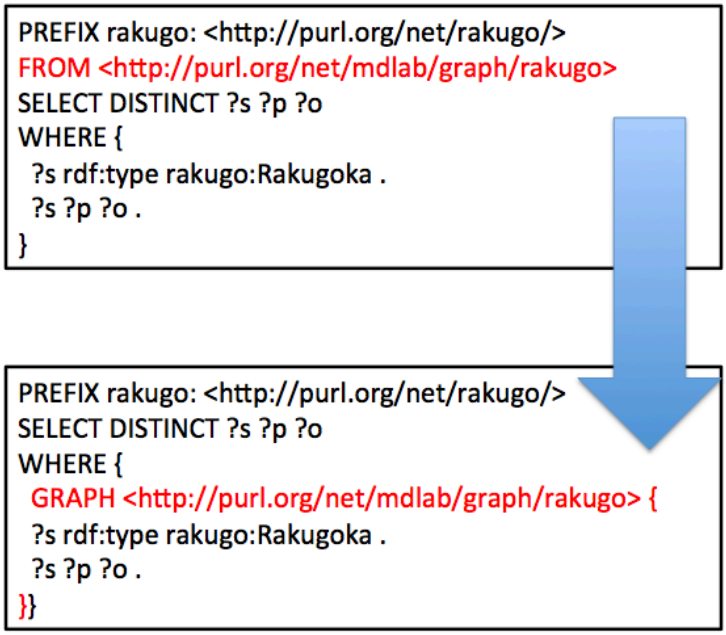


図 42 SPARQL クエリで使えない構文への対応例

4) トリプル数が多い、大きなデータセットのスキーマ推定に時間がかかってしまう

評価実験では公開されている RDF ファイルもしくは SPARQL Endpoint で、トリプル数がおよそ 50000 件を超えたものは実験中に推定が完了しなかった。しかし、分類 C において、RDF ファイルでは推定が完了しなかったエントリーでも SPAQRL Endpoint に対して推定を行うと推定が完了する場合があった。これは機械の計算能力が原因と思われる。この解決には、本推定システムを計算能力の高いサーバ上で実行する他に、公開されているデータセットのインスタンス全件を用いている手法を改善することで、アルゴリズムを改善することが求められる。

5) データセット公開者が推奨としたいタームがあってもインスタンスがなければ抽出できない

スキーマが公開されているデータセットで本システムを実行したところ、公開されているスキーマで定義されている項目が抽出できていなかった。推定できなかった項目は、スキーマを設計する際に「情報があれば記述してほしい」といった目的で定義された項目であると思われる。例えば現役アイドルの活動状況を LOD 化する際にスキーマ設計者が「引退年」という項目を定義しても、データセット作成時に引退者がいなければデータセット中にはその項目は記述されない。記述されていない項目については本手法では抽出できない。しかし、データセットで記述されていない項目は SPARQL クエリで問い合わせても取得できないので、LOD 利用者が LOD データセットを活用する上では実用上十分なスキーマが推定される。

8. 関連研究

本研究の「LOD を利用しやすくする」という研究目的、対象が類似する研究として、市瀬龍太郎らは、LOD データセットのマニュアルとして、LOD 全体の大きなオントロジーから、データセット内でコアとなるクラス、プロパティを TF-IDF を参考にした計算式から求め、結びつけ、LOD 全体を対象とするオントロジーを構築する手法を提案している[35]。LOD の構造を把握することを支援するという目的は類似しているが、利用したいデータセット単体を対象とするメタデータスキーマを推定する本研究とは対象が異なる。

Ruben Menders らはアプリケーション内で利用されている XML データのスキーマを XSLT スタイルシートから抽出する手法を提案している[38]。XSLT スタイルシートには変換後の XML データに用いられている名前空間宣言や構造が記述されている。本研究ではスキーマの推定にデータセット内のインスタンスを用いているのに対し、関連研究では XML データを指定の構造に変換するための XSLT スタイルシートを用いて構造を推定している点が異なる。

9. おわりに

本論文では LOD データセットの利用性向上のためにメタデータスキーマが公開されていない LOD データセットのメタデータスキーマを推定することを目的として、その推定手法を提案し、その評価を行った。

LOD データセットに対して構造化問い合わせを行う際に、LOD データセットのメタデータ語彙定義とメタデータ記述規則を記述したメタデータスキーマがあると便利である。本研究ではメタデータスキーマが公開されていない LOD データセットのメタデータスキーマを、そのデータセットのメタデータインスタンスから推定した。このとき、メタデータ語彙定義や Web オントロジーから推論することを前提としたデータセットでは明確にリソースのクラスなどが記述されないことが想定されたので、それらを既存メタデータスキーマから取得することでスキーマの推定に必要な情報を補足している。

提案手法をシステムとして実装し、LOD データセットに対して実行したところ、いつくかの場合にスキーマ推定が行えなかった。これは、データセットのスキーマ設計が公開されているスキーマに従っていないもの、SPARQL Endpoint の公開環境の違いよるもの、データセットの蓄積トリプル数が大きいものなどがあることが原因であった。今後は、公開されているスキーマに従っていないデータセットかどうか、SPARQL Endpoint の公開環境はどうなっているかの判断手法と、推定アルゴリズムの軽量化が求められる。

謝辞

これまでの三年間の研究を進めるにあたり、研究の進め方から論文の構成、まとめ方など、様々な場面でご指導、ご教授いただいた杉本重雄先生と永森光晴先生の両先生に感謝を申し上げます。併せて、本研究の研究対象や解決すべき課題の発見などについて多くのご意見をくださった森嶋厚行先生と阪口哲男先生に感謝いたします。

また、本間維先輩をはじめとする杉本・永森研究室の皆様にも数多くの助言を頂きました。

ここに、心より感謝の意を表します。

参考文献

[1] Linked Data -Connect Distributed Data across the Web. http://linkeddata.org (参照 2014-2-19)

[2] The Linking Open Data cloud diagram. http://lod-cloud.net (参照 2014-2-19)

[3] 西出頼継,本間維,永森光晴,杉本重雄.” 日本の Open Data 活用を目的としたデータセットのスキーマ分析とリンク関係の調査”. 情報処理学会第 112 回情報基礎とアクセス技術研究会 (IFAT),2013.

[4] メタデータ情報基盤事業. http://www.meta-proj.jp/index.html (参照 2014-2-19)

[5] 平成 22 年度新 ICT 利活用サービス創出支援事業(電子出版の環境整備)メタデータ情報基盤構築事業報告書. http://www.meta-proj.jp/metaproj2010.pdf (参照 2014-2-19)

[6] Tim Berners-Lee, “W3 future directions”,Plenary at WWW Geneva 94, 1994-09; http://www.w3.org/Talks/WWW94Tim/ (参照 2014-2-19)

[7] Tim Berners-Lee, “Semantic Web Road map”, 1998,http://www.w3.org/DesignIssues/Semantic.html (参照 2014-2-19)

[8] DBpedia . http://dbpedia.org/ (参照 2014-2-19)

[9] Linked Open Data Initiative.「日本語版 Linked Data クラウド図」 http://linkedopendata.jp/?p=411 (参照 2014-2-19)

[10] Yokohama Art Spot. http://lod.ac/apps/yas/ (参照 2014-2-19)

[11] ヨコハマ・アート・LOD | 公益財団法人 横浜市芸術文化振興財団. http://fp.yafjp.org/yokohama_art_lod (参照 2014-2-19)

[12] PinQA (ピンカ) - 行きたいリストが作れるおでかけマガジン http://pinqa.com/ (参照 2014-2-19)

[13] LODAC Project. http://lod.ac (参照 2014-2-19)

[14] ”税金はどこへ行った？ - WHERE DOES MY MONEY GO?” http://spending.jp/ (参照 2014-2-19)

[15] Linked Open Data Challenge Japan 2012, ”Where Does My Money Go? 日本語版”. http://lod.sfc.keio.ac.jp/challenge2012/show_status.php?id=a023 (参照 2014-2-19)

[16] RDF - Semantic Web Standards. http://www.w3.org/RDF/ (参照 2014-2-19)

[17] 神崎正英. “RDF の概念とモデル.” セマンティック・ウェブのための RDF/OWL 入門. 森北出版株式会社,2005, p11-23.

[18] Virtuoso Universal Server. http://virtuoso.openlinksw.com/ (参照 2014-2-19)

[19] openRDF.org:Home. http://www.openrdf.org/ (参照 2014-2-19)

[20] semanticweb.org:SPARQL endpoint . http://semanticweb.org/wiki/SPARQL_endpoint (参照 2014-2-19)

[21] SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/ (参照 2014-2-19)

[22] SPARQL 1.1 Query Language. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (参照 2014-2-19)

[23] 第 2 回 Linked Data 勉強会 発表資料. http://linkeddata.jp/2011/07/24/lodjp-contents/ (参照 2014-2-19)

[24] 落語家 LOD. http://mdlab.slis.tsukuba.ac.jp/lodc2012/rakugo/ (参照 2014-2-19)

[25] FOAF Vocabulary Specification 0.98. http://xmlns.com/foaf/spec/ (参照 2014-2-19)

[26] Milael Nilsson. Description Set Profiles: A constraint language for Dublin Core Application Profiles. DCMI Working Draft, 03 2008. http://dublincore.org/documents/dc-dsp/ (参照 2014-2-19)

[27] Paul V. Biron and ashok Malhotra eds., “XML Schema Part 2: Datatypes”, 2001-05-02, W3C Recommendation, http://www.w3.org/TR/xmlschema-2/ (参照 2014-2-19)

[28] Linked Open Vocabularies. http://labs.mondeca.com/dataset/lov/index.html (参照 2014-2-19)

[29] メタデータ基盤システム. https://www.metabridge.jp/ (参照 2014-2-19)

[30] トム・ヒース, クリスチャン・バイツァー 著,武田英明 監訳.「Linked Data Web をグローバルなデータ空間にする仕組み」,2013.

[31] CKAN 日本語. http://data.linkedopendata.jp/ (参照 2014-2-19)

[32] Open Data METI. http://datameti.go.jp/data/ (参照 2014-2-19)

[33] Linked Open Data チャレンジ Japan . http://lod.sfc.keio.ac.jp/ (参照 2014-2-19)

[34] Linked Open Data チャレンジ Japan 2013. http://lod.sfc.keio.ac.jp/challenge2013/ (参照 2014-1-10)

[35] Lihua Zhao and Ryutaro Ichise. “Instance-based ontological knowledge acquisition.” eswc2013, 2013.

[36] メタデータ情報基盤構築事業.メタデータ情報共有のためのガイドライン. http://www.soumu.go.jp/main_content/000132512.pdf (参照 2014-2-19)

[37] SPARQL 1.1 Service Description. http://www.w3.org/TR/sparql11-service-description/

[38] Ruben Mendes, Jose Borbinha, Hugo Manguinhas. “Extractiong Output Schemas from XSLT Stylesheets and Their Possible Applications” dcd2012, 2012.

[39] ねじ LOD. http://monodzukurilod.org/neji/ (参照 2014-2-19)

[40] RDF/XML Syntax Specification. http://www.w3.org/TR/rdf-syntax-grammar/ (参照 2014-2-19)

[41] N-Triples. http://www.w3.org/TR/2013/NOTE-n-triples-20130409/ (参照 2014-2-19)

[42] RDF 1.1 Turtle. http://www.w3.org/TR/turtle/ (参照 2014-2-19)

[43] gkellogg. “Ruby-rdf/sparql”. GitHub. https://github.com/ruby-rdf/sparql (参照 2014-2-19)

[44] SPARQL Client for RDF.rb . http://ruby-rdf.github.io/sparql-client/ (参照 2014-2-19)